

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-325759

(43) 公開日 平成9年(1997)12月16日

(51) Int.Cl. ⁸	識別記号	庁内整理番号	F I	技術表示箇所
G 0 9 G 5/36	5 3 0		G 0 9 G 5/36	5 3 0 Z
A 6 3 F 9/22			A 6 3 F 9/22	B
				H
G 0 6 T 11/00			G 1 0 K 15/04	3 0 2 G
G 1 0 K 15/04	3 0 2		G 0 6 F 15/72	A
審査請求 未請求 請求項の数173 書面 外国語出願 (全 403 頁)				

(21) 出願番号 特願平8-352115

(22) 出願日 平成8年(1996)11月22日

(31) 優先権主張番号 08/561718

(32) 優先日 1995年11月22日

(33) 優先権主張国 米国 (US)

(71) 出願人 000233778

任天堂株式会社

京都府京都市東山区福稲上高松町60番地

(71) 出願人 596016535

シリコン グラフィックス インコーポレ
イテッド

Silicon Graphics, In
c.

アメリカ合衆国 94039 カリフォルニア
州 マウンテン ビュー ノース ショー
アライン プールバード 2011

(74) 代理人 弁理士 小笠原 史朗

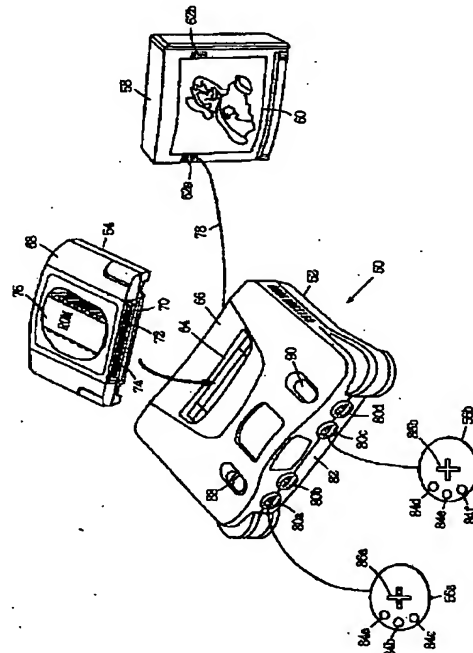
最終頁に続く

(54) 【発明の名称】 高速高効率3Dグラフィックス及びデジタル音声信号処理を提供するコプロセッサを備える高性能低コストビデオゲームシステム

(57) 【要約】

【課題】 多くのユーザが支払い可能な範囲内にある低コストであって、3Dでワールドをモデル化し、かつ変更可能な視点に基づいて選択された2次元ビューイング平面上に当該モデルを投影できるという高性能3次元グラフィックスシステムである。

【解決手段】 上記視点は、ゲームコントローラ等によるユーザの入力操作を制御することにより、インタラクティブにリアルタイムで変更可能である。そして、これに対応する変更画像をカラーテレビ受像機の画面上に迅速に生成させる。本発明は、上記内容を実現するためのグラフィックス/音声コプロセッサの構造および動作、具体的には、コプロセッサの内部構造、グラフィックス/音声出力のために実行する処理、外部の世界と通信する方法、単一化されたRAM、および上記コプロセッサを制御するためにメインプロセッサがコプロセッサに送るコマンドおよびそのフォーマットについて述べている。



【特許請求の範囲】

【請求項1】 対話式ユーザ入力装置と、

前記入力装置に接続され、アドレス空間を有し、前記ユーザ入力装置からの入力に応じて対話式に視点を選択するメインプロセッサと、

前記メインプロセッサに接続され、3次元ワールドを表す多角形を2次元ビューイング平面上に投影することにより、選択された視点に応じて画像データを対話式に生成するために所定のグラフィックス特徴セットを提供し、

少なくともグラフィックス機能とオーディオ処理機能とに共用され、スカラユニットおよび複数の計算を並列に実行できるベクトルユニットとを有し、マイクロコードを記憶するマイクロコード記憶装置を有し、当該グラフィックス機能と当該オーディオ処理機能を実行するために、当該マイクロコード記憶装置中の当該マイクロコードを実行する信号プロセッサと、

第1の部分は、カラーインデックスされたテクスチャマップとカラーインデックスされていないテクスチャマップとを記憶でき、第2の部分は、テクスチャマップおよび/またはカラーインデックスされたテクスチャマップ用のカラーlookupテーブルを記憶できるテクスチャメモリを有し、サイクル毎1画素モードあるいはサイクル毎2画素モードを与えてハードウェアを最少にするとともに、詳細処理のレベルを含む豊富な特徴セットを与える表示パイプラインハードウェアを含む表示プロセッサと、

ビデオインタフェイスと、

音声インタフェイスと、

シリアルインタフェイスと、

パラレル周辺インタフェイスとを含むコプロセッサと、

前記信号プロセッサ、前記表示プロセッサ、前記ビデオインタフェイス、前記音声インタフェイス、前記シリアルインタフェイス、前記パラレル周辺インタフェイスの各々は、メインメモリにアクセスする回路を備え、9ビットワイドバスを介して前記コプロセッサに接続され、前記コプロセッサと前記メインプロセッサとに共通のアドレス空間を与え、更に少なくとも、

前記メインメモリにより実行される命令と、

カラーフレームバッファと、

奥行バッファと、

グラフィックマイクロコードと、

音声処理マイクロコードと、

少なくとも一つの表示リストと、

少なくとも一つのテクスチャマップと、

少なくとも一つのオーディオ出力バッファのデータ構造とを記憶するメインメモリと、

前記コプロセッサの前記ビデオインタフェイスに接続され、カラーテレビ受像機上に表示されるビデオ信号を生成するビデオ信号生成回路と、

ハウジング、セキュリティチップ、読取り専用メモリと、少なくとも一つの別のメモリ装置を有する取外し可能記憶装置と、当該読取り専用メモリおよび当該別のメモリ装置を、前記メインプロセッサのアドレス空間にマッピングする構造を備える前記コプロセッサと、初期に前記グラフィックスおよびオーディオ処理マイクロコードを記憶する前記読取り専用メモリと、前記コプロセッサを前記取外し可能記憶装置に接続するコネクタと、

10 前記コプロセッサのシリアルインタフェイスに接続され、シリアルインタフェイス機能とセキュリティ機能を実行するプロセッサ、およびメインプロセッサ初期プログラムロード命令を与えるブートROMを備え、前記コネクタを介して前記取外し可能記憶装置のセキュリティチップに接続されているシリアル周辺インタフェイス回路とからなる対話式ビデオゲームシステム。

【請求項2】 少なくとも一つのユーザ入力装置と、共通のアドレス空間を与えるメインランダムアクセスメモリと、

20 前記メインメモリをアドレス指定するために接続され、かつ、前記ユーザ入力装置にも接続され、前記ユーザ入力装置から入力された入力にリアルタイムで応答して前記メインメモリに命令を記憶し、かつ、前記メインメモリからの命令を実行し、また、少なくともグラフィックスコマンドの表示リストおよびオーディオコマンドの再生リストを前記メインメモリに記憶するメインプロセッサと、

30 前記メインメモリをアドレス指定するために接続され、前記メインメモリに記憶されたマイクロコードを取出し、かつ実行し、前記表示リストのグラフィックスコマンドおよび前記再生リストのオーディオコマンドを前記メインメモリから読み出し、前記再生リストのオーディオコマンドに回答してオーディオサンプルデータを生成し、かつ、前記表示リストに回答してグラフィックス表示コマンドを生成し、前記メインメモリ内に割当てられたオーディオ出力バッファに前記サンプルデータを記憶する信号プロセッサと、

40 前記メインメモリをアドレス指定するために接続され、前記メインメモリに記憶された少なくとも一つのテクスチャマップおよび他のグラフィックスデータに少なくとも部分的に基づいて画像データを生成し、グラフィックス表示コマンドに回答して当該画像データを作成し、当該画像データを前記メインメモリ内のカラー画像フレームバッファ内に記憶する表示プロセッサと、

前記メインメモリをアドレス指定するために接続され、表示ラスタスキャンに同期して、前記カラー画像フレームバッファを読み取るビデオインタフェイスと、

50 前記メインメモリをアドレス指定するために接続され、リアルタイム音響生成に同期して、前記オーディオ出力バッファを読み取るオーディオインタフェイスとからなる

る対話式リアルタイムグラフィックス表示システム。

【請求項3】 メインプロセッサと、当該メインプロセッサに接続されたコプロセッサと、当該コプロセッサに接続され当該メインプロセッサおよび当該コプロセッサによりアドレス指定できるメインランダムアクセスメモリと、および表示用ビデオ信号を生成するビデオ信号生成構造とを有するグラフィックス表示システムを動作する方法であって、

(a) メインプロセッサコードを前記メインメモリに記憶するステップと、

(b) コプロセッサコード、タスクリスト、少なくとも1つのテクスチャマップ、およびカラーlookupアップテーブルを前記メインメモリに記憶するステップを含み、前記記憶するステップにより記憶された前記メインプロセッサコードを前記メインプロセッサにより実行するステップと、

(c) 前記メインメモリから前記タスクリストを取り出すステップと、

(d) 前記ステップ(b)により記憶された前記コプロセッサコードに、少なくとも部分的にしたがって、前記コプロセッサにより前記タスクリストを処理し、以下の、(1) 前記テクスチャマップおよび前記カラーlookupアップテーブルを、前記メインメモリからオンチップテクスチャメモリにロードすること、(2) スカラユニットおよびベクトルユニットに平行して複数の計算を実行することを含むベクトルユニットを用いて、1組の頂点について少なくとも1つの3次元幾何変換を実行すること、(3) 前記3次元幾何変換に基づいて三角コマンドを生成すること、(4) 前記三角コマンドにตอบสนองして画素値を生成すること、(5) 前記テクスチャメモリを2度アクセスして、前記三角コマンドに基づいてカラーインデックステクセルを出力すること、(6) 前記テクセルと生成された画素値とを組み合わせ、複合画素値を生成すること、(7) 前記メインメモリに記憶されたフレームバッファ内の画素値をアクセスすること、(8) 前記フレームバッファ内に記憶された少なくとも1つの画素値により、前記複合画素値をブレンドすること、(9) 前記メインメモリに記憶された奥行バッファを用いた比較に基づいて、前記複合画素値を前記フレームバッファに条件付きで書き込むこと、(10) 前記ベクトルユニットに平行して複数の計算を実行することを含み、前記スカラユニットおよび前記ベクトルユニットを用いて出力オーディオサンプルを生成すること、(11) 前記出力オーディオサンプルを前記メインメモリに記憶すること、の各ステップの実行をも含む処理ステップと、

(e) カラーテレビ受像機の走査に同期してリアルタイムで前記フレームバッファを読みとり、かつ、前記フレームバッファの内容を複合ビデオ信号に変換するステップと、

(f) 前記記憶された出力オーディオサンプルをリアルタイムで読み取り、かつ、前記オーディオサンプルをステレオ音響に変換するステップと、

を備えた、グラフィックス表示システムの動作方法。

【請求項4】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示モード制御コマンドを生成するプロセスであって、少なくとも1つのセットモードコマンドを生成するステップを含み、当該セットモードコマンドは、

10 101111の6ビットバイナリ値を含むコマンド識別子フィールドと、

下記のモード制御フィールド、つまり、

(k) 次のプリミティブを読み出す前にフレームバッファにプリミティブを書き込ませるかどうかを特定する、アトミックプリミティブモードフィールド、

(i) 表示パイプラインサイクル制御モードを選択する、サイクルタイプモードフィールド、

(h) 遠近法テクスチャ修正を選択的に可能にする、遠近法テクスチャイネーブルモードフィールド、

20 (g) テクスチャの詳細な処理を選択的に可能にする、テクスチャ詳細モードフィールド、

(f) テクスチャを鮮明にすることを選択的に可能にする、テクスチャ鮮明イネーブルモード、

(e) テクスチャの詳細レベル値の処理を選択的に可能にする、テクスチャ詳細イネーブルモード、

(d) カラーlookupアップテーブルからテクスチャ値を選択的に検索することを可能にする、イネーブルlookupアップテーブルモードフィールド、

30 (c) カラーlookupアップテーブル中のテクセルのタイプを特定する、テクスチャlookupアップテーブルタイプモードフィールド、

(b) テクセルがどのようにサンプリングされるべきかを特定する、サンプリングタイプモードフィールド、

(a) テクセルが2×2ハーフテクセル補間を用いてフィルタリングされるべきかどうかを特定する、ミッドテクセルモードフィールド、

40 (Z) テクスチャフィルタがパイプラインサイクル0中のテクセルをバイリニア的に補間すべきかどうかを特定する、第1のバイラップモードフィールド、

(Y) テクスチャフィルタがパイプラインサイクル1中のテクセルをバイリニア的に補間すべきかどうかを特定する、第2のバイラップモードフィールド、

(X) パイプラインサイクル0の間にテクスチャフィルタから出力されるテクセルが色変換されるべきかどうかを特定する、テクセル変換モードフィールド、

(W) クロマキーボード入力を選択的に可能にする、クロマキーイネーブルモードフィールド、

(V2) RGBディザリングのタイプを選択する、RGBディザ選択モードフィールド、

50 (V1) アルファディザリングのタイプを選択する、ア

ルファディザ選択モードフィールド、

(V) ブレンドパラメータを特定する、複数のブレンドモードワード、

(M) 前記ブレンドが強制使用可能にされるべきかどうかを特定する、強制ブレンドイネーブルモードフィールド、

(L) 有効範囲が画素アルファを決定するために用いられるべきかどうかを特定する、アルファ有効範囲選択モードフィールド、

(K) アルファによって乗算された有効範囲が画素アルファおよび有効範囲を決定するために用いられるべきかどうかを特定する、有効範囲タイムスアルファ選択モードフィールド、

(J) Zバッファリングを特定する、Zモード選択モードフィールド、

(I) 有効範囲宛先を特定する、有効範囲宛先モードフィールド、

(H) 色が有効範囲からはみ出したところでのみ更新されるべきかどうかを特定する、有効範囲のカラーモードフィールド、

(G) 色および／または有効範囲読み出し／修正／書き込みフレームバッファメモリアクセスを選択的に可能にする、画像読み出しイネーブルモードフィールド、

(F) 色書き込みが可能にされるかどうかによってZバッファ書き込みを選択的に可能にする、Zアップデートイネーブルモードフィールド、

(E) 実行比較において可能な条件付色書き込みを特定する、Z比較イネーブルモードフィールド、

(D) 有効範囲を用いたブレンドイネーブルを許可する、アンチエリアスイネーブルモードフィールド、

(C) プリミティブ実行と画素実行の間で選択をする、Zソース選択モードフィールド、

(B) ランダムノイズがアルファ比較に用いられるべきかどうかを特定する、ディザアルファイネーブルモードフィールド、

(A) アルファ比較上で条件付色書き込みを可能とする、アルファ比較イネーブルモードフィールド、
のうちの少なくとも1つを有する、プロセス。

【請求項5】 画素毎の1サイクルモード、画素毎の2サイクルモード、コピーモード、およびフィルモードの表示パイプラインサイクル制御モードを選択するサイクルタイプモードを生成するステップを含む、請求項4に記載のプロセス。

【請求項6】 (1) 5ビットの赤、5ビットの緑、5ビットの青、および1ビットのアルファのRGBAフォーマット内のカラールックアップテーブル中におけるテクセルの記憶、

(2) 8ビット輝度値および8ビットアルファ値を供給する輝度アルファフォーマット内のカラールックアップテーブル中におけるテクセルの記憶、

のうちのいずれかを選択するテクスチャルックアップタイプモードフィールドを生成するステップを含む、請求項4に記載のプロセス。

【請求項7】 (1) ポイントサンプリング、

(2) 2×2配列サンプリング、

のうちのいずれかを選択をするサンプリングタイプモードフィールドを生成するステップを含む、請求項4に記載のプロセス。

【請求項8】 (1) マジックスクエアマトリックス、

(2) ベイヤーマトリックス、

(3) ノイズ、

(4) ディザリング無し、

に基づくディザリングのうちのいずれかを選択するRGBディザ選択モードフィールドを生成するステップを含む、請求項4に記載のプロセス。

【請求項9】 (1) 所定のパターン、

(2) 所定のパターンのネガティブ、

(3) ノイズ、

(4) ディザリング無し、

20 に基づくディザリングを特定するアルファディザリング選択モードフィールドを生成するステップを含む、請求項4に記載のプロセス。

【請求項10】 パイプラインサイクル0の間、第1のブレンド入力を選択的な増大、

パイプラインサイクル1の間、第1のブレンド入力の選択的な増大、

パイプラインサイクル0の間、第2のブレンド入力の選択的な増大、

パイプラインサイクル1の間、第2のブレンド入力の選択的な増大、

30 パイプラインサイクル0の間、第3のブレンド入力の選択的な増大、

パイプラインサイクル1の間、第3のブレンド入力の選択的な増大、

パイプラインサイクル0の間、第4のブレンド入力の選択的な増大、

パイプラインサイクル1の間、第4のブレンド入力の選択的な増大、

を特定するブレンドパラメータを特定する複数のブレンドモードワードを生成するステップを含む、請求項4に記載のプロセス。

【請求項11】 (1) クランプ、

(2) ラップ、

(3) 全有効範囲への作用、

(4) 保存、

のうちのいずれかの有効範囲宛先モードを選択する有効範囲宛先モードフィールドを生成するステップを含む、請求項4に記載のプロセス。

【請求項12】 (1) 不透明、

50 (2) 浸透、

(3) 透明、

(4) デカル、

のうちのいずれかのZバッファリングモードを選択するZモード選択モードフィールドを生成するステップを含む、請求項4に記載のプロセス。

【請求項13】 3次元グラフィックスシステムによって処理するための、少なくとも1つの3次元表示モード制御コマンドを生成するシステムであって、少なくとも1つのプロセッサと、少なくとも1つのメモリと、少なくとも1つのセットモードコマンドを供給するために、前記プロセッサと前記メモリに結合される回路と、を含み、前記セットモードコマンドは、101111の6ビットバイナリ値を含むコマンド識別子フィールドと、

下記のモード制御フィールド、つまり、

(k) 次のプリミティブを読み取る前にフレームバッファにプリミティブを書き込ませるかどうかを特定する、アトミックプリミティブモードフィールド、

(i) 表示パイプラインサイクル制御モードを選択する、サイクルタイプモードフィールド、

(h) 遠近法テクスチャ修正を選択的に可能にする、遠近法テクスチャイネーブルモードフィールド、

(g) テクスチャの詳細な処理を選択的に可能にする、テクスチャ詳細モードフィールド、

(f) テクスチャを鮮明にすることを選択的に可能にする、テクスチャ鮮明イネーブルモード、

(e) テクスチャの詳細レベル値の処理を選択的に可能にする、テクスチャ詳細イネーブルモード、

(d) カラーlookupアップテーブルからテクスチャ値を選択的に検索することを可能にする、イネーブルlookupアップテーブルモードフィールド、

(c) カラーlookupアップテーブル中のテクセルのタイプを特定する、テクスチャlookupアップテーブルタイプモードフィールド、

(b) テクセルがどのようにサンプリングされるべきかを特定する、サンブルタイプモードフィールド、

(a) テクセルが2×2ハーフテクセル補間を用いてフィルタリングされるべきかどうかを特定する、ミッドテクセルモードフィールド、

(Z) テクスチャフィルタがパイプラインサイクル0中のテクセルをバイリニア的に補間すべきかどうかを特定する、第1のバイラップモードフィールド、

(Y) テクスチャフィルタがパイプラインサイクル1中のテクセルをバイリニア的に補間すべきかどうかを特定する、第2のバイラップモードフィールド、

(X) パイプラインサイクル0の間にテクスチャフィルタから出力されるテクセルが色変換されるべきかどうかを特定する、テクセル変換モードフィールド、

(W) クロマキーボード入力を選択的に可能にする、ク

ロマキーイネーブルモードフィールド、

(V2) RGBディザリングのタイプを選択する、RGBディザ選択モードフィールド、

(V1) アルファディザリングのタイプを選択する、アルファディザ選択モードフィールド、

(V) ブレンダパラメータを特定する、複数のブレンドモードワード、

(M) 前記ブレンダが強制使用可能にされるべきかどうかを特定する、強制ブレンドイネーブルモードフィールド、

10 (L) 有効範囲が画素アルファを決定するために用いられるべきかどうかを特定する、アルファ有効範囲選択モードフィールド、

(K) アルファによって増大された有効範囲が画素アルファおよび有効範囲を決定するために用いられるべきかどうかを特定する、有効範囲タイムスアルファ選択モードフィールド、

(J) Zバッファリングを特定する、Zモード選択モードフィールド、

20 (I) 有効範囲宛先を特定する、有効範囲宛先モードフィールド、

(H) 色が有効範囲からはみ出したところでのみ更新されるべきかどうかを特定する、有効範囲のカラーモードフィールド、

(G) 色および/または有効範囲読み出し/修正/書き込みフレームバッファメモリアクセスを選択的に可能にする、画像読み出しイネーブルモードフィールド、

(F) 色書き込みが可能にされるかどうかによってZバッファ書き込みを選択的に可能にする、Zアップデートイネーブルモードフィールド、

(E) 実行比較において可能な条件付色書き込みを特定する、Z比較イネーブルモードフィールド、

(D) 有効範囲を用いたブレンドイネーブルを許可する、アンチエリアスイネーブルモードフィールド、

(C) プリミティブ実行と画素実行の間で選択をする、Zソース選択モードフィールド、

(B) ランダムノイズがアルファ比較に用いられるべきかどうかを特定する、ディザアルファイネーブルモードフィールド、

40 (A) アルファ比較上で条件付色書き込みを可能とする、アルファ比較イネーブルモードフィールド、

のうちの少なくとも1つを有する、システム。

【請求項14】 画素毎の1サイクルモード、画素毎の2サイクルモード、コピーモード、およびフィルモードの表示パイプラインサイクル制御モードを選択するサイクルタイプモードを生成する手段を含む、請求項13に記載のシステム。

【請求項15】 (1) 5ビットの赤、5ビットの緑、5ビットの青および1ビットのアルファのRGBAフォーマット内のカラーlookupアップテーブル中におけるテ

クセルの記憶、

(2) 8ビット輝度値および8ビットアルファ値を供給する輝度アルファフォーマット内のカラーlookupアップテーブル中におけるテクセルの記憶、
のうちのいずれかを選択するテクスチャlookupアップタイプモードフィールドを供給する手段を含む、請求項13に記載のシステム。

【請求項16】 (1) ポイントサンプリング、

(2) 2×2配列サンプリング、

のうちのいずれかを選択するサンプリングタイプモードフィールドを供給する回路を含む、請求項13に記載のシステム。

【請求項17】 (1) マジックスクエアマトリックス、

(2) ベイヤーマトリックス、

(3) ノイズ、

(4) ディザリング無し、

に基づくディザリングのうちのいずれかを選択するRGBディザ選択モードフィールドを供給する回路を含む、請求項13に記載のシステム。

【請求項18】 (1) 所定のパターン、

(2) 所定のパターンのネガティブ、

(3) ノイズ、

(4) ディザリング無し、

に基づくディザリングを特定するアルファディザリング選択モードフィールドを生成する手段を含む、請求項13に記載のシステム。

【請求項19】 バイブラインサイクル0の間、第1のブレンダ入力の選択的な増大、

バイブラインサイクル1の間、第1のブレンダ入力の選択的な増大、

バイブラインサイクル0の間、第2のブレンダ入力の選択的な増大、

バイブラインサイクル1の間、第2のブレンダ入力の選択的な増大、

バイブラインサイクル0の間、第3のブレンダ入力の選択的な増大、

バイブラインサイクル1の間、第3のブレンダ入力の選択的な増大、

バイブラインサイクル0の間、第4のブレンダ入力の選択的な増大、

バイブラインサイクル1の間、第4のブレンダ入力の選択的な増大、

を特定するブレンダパラメータを特定する複数のブレンダモードワードを生成する手段を含む、請求項13に記載のシステム。

【請求項20】 (1) クランプ、

(2) ラップ、

(3) 全有効範囲への作用、

(4) 保存、

のうちのいずれかの有効範囲宛先モードを選択する有効範囲宛先モードフィールドを生成する回路を含む、請求項13に記載のシステム。

【請求項21】 (1) 不透明、

(2) 浸透、

(3) 透明、

(4) デカル、

のうちのいずれかのZバッファリングモードを選択するZモード選択モードフィールドを供給する回路を含む、請求項13に記載のシステム。

【請求項22】 3次元グラフィックスシステムにおいて、少なくとも1つのセットモードコマンドを解釈するプロセスであって、

(1) 101111の6ビットバイナリ値を含むコマンド識別子フィールドを解釈するステップと、

(2) 下記のモード制御フィールド、つまり(k)次のプリミティブを読み取る前にフレームバッファにプリミティブを書き込ませるかどうかを特定する、アトミックプリミティブモードフィールド、(i)表示バイブラインサイクル制御モードを選択する、サイクルタイプモードフィールド、(h)遠近法テクスチャ修正を選択的に可能にする、遠近法テクスチャイネーブルモードフィールド、(g)テクスチャの詳細な処理を選択的に可能にする、テクスチャ詳細モードフィールド、

(f)テクスチャを鮮明にすることを選択的に可能にする、テクスチャ鮮明イネーブルモード、(e)テクスチャの詳細レベル値の処理を選択的に可能にする、テクスチャ詳細イネーブルモード、(d)カラーlookupアップテーブルからテクスチャ値を選択的に検索することを可能にする、イネーブルlookupアップテーブルモードフィールド、(c)カラーlookupアップテーブル中のテクセルのタイプを特定する、テクスチャlookupアップテーブルタイプモードフィールド、(b)テクセルがどのようにサンプリングされるべきかを特定する、サンプリングタイプモードフィールド、(a)テクセルが2×2ハーフテクセル補間を用いてフィルタリングされるべきかどうかを特定する、ミッドテクセルモードフィールド、(Z)テクスチャフィルタがバイブラインサイクル0中のテクセルをバイリニア的に補間すべきかどうかを特定する、第1のバイラップモードフィールド、

(Y)テクスチャフィルタがバイブラインサイクル1中のテクセルをバイリニア的に補間すべきかどうかを特定する、第2のバイラップモードフィールド、(X)バイブラインサイクル0の間にテクスチャフィルタから出力されるテクセルが色変換されるべきかどうかを特定する、テクセル変換モードフィールド、(W)クロマキーボード入力を選択的に可能にする、クロマキーイネーブルモードフィールド、(V2)RGBディザリングのタイプを選択する、RGBディザ選択モードフィールド、(V1)アルファディザリングのタイプを選択す

る、アルファディザ選択モードフィールド、(V) ブレンドパラメータを特定する、複数のブレンドモードワード、(M) 前記ブレンドが強制使用可能にされるべきかどうかを特定する、強制ブレンドイネーブルモードフィールド、(L) 有効範囲が画素アルファを決定するために用いられるべきかどうかを特定する、アルファ有効範囲選択モードフィールド、(K) アルファによって増大された有効範囲が画素アルファおよび有効範囲を決定するために用いられるべきかどうかを特定する、有効範囲タイムスアルファ選択モードフィールド、(J) Zバッファリングを特定する、Zモード選択モードフィールド、(I) 有効範囲宛先を特定する、有効範囲宛先モードフィールド、(H) 色が有効範囲からはみ出したところでのみ更新されるべきかどうかを特定する、有効範囲のカラーモードフィールド、(G) 色および/または有効範囲読み出し/修正/書き込みフレームバッファメモリアクセスを選択的に可能にする、画像読み出しイネーブルモードフィールド、(F) 色書き込みが可能にされるかどうかによってZバッファ書き込みを選択的に可能にする、Zアップデートイネーブルモードフィールド、(E) 実行比較において可能な条件付色書き込みを特定する、Z比較イネーブルモードフィールド、(D) 有効範囲を用いたブレンドイネーブルを許可する、アンチエイリアスイネーブルモードフィールド、(C) プリミティブ実行と画素実行の間で選択をする、Zソース選択モードフィールド、(B) ランダムノイズがアルファ比較に用いられるべきかどうかを特定する、ディザアルファイネーブルモードフィールド、(A) アルファ比較上で条件付色書き込みを可能とする、アルファ比較イネーブルモードフィールド、のうちの少なくとも1つを解釈するステップと、

(3) 少なくとも部分的に前記ステップ(2)に基づいて画像を生成するステップと、を含む、プロセス。

【請求項23】 画素毎の1サイクルモード、画素毎の2サイクルモード、コピーモード、およびフィルモードの表示パイプラインサイクル制御モードを選択するサイクルタイプモードを解釈するステップを含む、請求項22に記載のプロセス。

【請求項24】 (1) 5ビットの赤、5ビットの緑、5ビットの青、および1ビットのアルファのRGBAフォーマット内のカラールックアップテーブル中におけるテクセルの記憶、

(2) 8ビット輝度値および8ビットアルファ値を供給する輝度アルファフォーマット内のカラールックアップテーブル中におけるテクセルの記憶、

のうちのいずれかを選択するテクスチュアルックアップタイプモードフィールドを解釈するステップを含む、請求項22に記載のプロセス。

【請求項25】 (1) ポイントサンプリング、

(2) 2×2配列サンプリング、
のうちのいずれかを選択をするサンプラタイプモードフィールドを解釈するステップを含む、請求項22に記載のプロセス。

【請求項26】 (1) マジックスクエアマトリックス、

(2) ベイヤーマトリックス、

(3) ノイズ、

(4) ディザリング無し、

10 に基づくディザリングのうちのいずれかを選択するRGBディザ選択モードフィールドを解釈するステップを含む、請求項22に記載のプロセス。

【請求項27】 (1) 所定のパターン、

(2) 所定のパターンのネガティブ、

(3) ノイズ、

(4) ディザリング無し、

に基づくディザリングを特定するアルファディザリング選択モードフィールドを解釈するステップを含む、請求項22に記載のプロセス。

20 【請求項28】 パイプラインサイクル0の間、第1のブレンド入力を選択的に増大、

パイプラインサイクル1の間、第1のブレンド入力の選択的な増大、

パイプラインサイクル0の間、第2のブレンド入力の選択的な増大、

パイプラインサイクル1の間、第2のブレンド入力の選択的な増大、

パイプラインサイクル0の間、第3のブレンド入力の選択的な増大、

30 パイプラインサイクル1の間、第3のブレンド入力の選択的な増大、

パイプラインサイクル0の間、第4のブレンド入力の選択的な増大、

パイプラインサイクル1の間、第4のブレンド入力の選択的な増大、

を特定するブレンドパラメータを特定する複数のブレンドモードワードを解釈するステップを含む、請求項22に記載のプロセス。

【請求項29】 (1) クランプ、

(2) ラップ、

(3) 全有効範囲への作用、

(4) 保存、

のうちのいずれかの有効範囲宛先モードを選択する有効範囲宛先モードフィールドを解釈するステップを含む、請求項22に記載のプロセス。

【請求項30】 (1) 不透明、

(2) 浸透、

(3) 透明、

(4) デカル、

50 のうちのいずれかのZバッファリングモードを選択する

Zモード選択モードフィールドを解釈するステップを含む、請求項22に記載のプロセス。

【請求項31】 少なくとも101111の6ビットバイナリ値を含むコマンド識別子フィールドを有する、少なくとも1つのセットモードコマンドを解釈する3次元グラフィックスシステムであって、

101111の6ビットバイナリ値を含むコマンド識別子フィールドを解釈する第1の復号器と、

(k) 次のプリミティブを読み取る前にフレームバッファにプリミティブを書き込ませるかどうかを特定する、アトミックプリミティブモードフィールド、を解釈する回路と、

(i) 表示パイプラインサイクル制御モードを選択する、サイクルタイプモードフィールド、を解釈する回路と、

(h) 遠近法テクスチャ修正を選択的に可能にする、遠近法テクスチャイネーブルモードフィールド、を解釈する回路と、

(g) テクスチャの詳細な処理を選択的に可能にする、テクスチャ詳細モードフィールド、を解釈する回路と、

(f) テクスチャを鮮明にすることを選択的に可能にする、テクスチャ鮮明イネーブルモード、を解釈する回路と、

(e) テクスチャの詳細レベル値の処理を選択的に可能にする、テクスチャ詳細イネーブルモード、を解釈する回路と、

(d) カラーlookupテーブルからテクスチャ値を選択的に検索することを可能にする、イネーブルlookupテーブルモードフィールド、を解釈する回路と、

(c) カラーlookupテーブル中のテクセルのタイプを特定する、テクスチャlookupテーブルタイプモードフィールド、を解釈する回路と、

(b) テクセルがどのようにサンプリングされるべきかを特定する、サンプリングタイプモードフィールド、を解釈する回路と、

(a) テクセルが2×2ハーフテクセル補間を用いてフィルタリングされるべきかどうかを特定する、ミッドテクセルモードフィールド、を解釈する回路と、

(Z) テクスチャフィルタがパイプラインサイクル0中のテクセルをバイリニア的に補間すべきかどうかを特定する、第1のバイラップモードフィールド、を解釈する回路と、

(Y) テクスチャフィルタがパイプラインサイクル1中のテクセルをバイリニア的に補間すべきかどうかを特定する、第2のバイラップモードフィールド、を解釈する回路と、

(X) パイプラインサイクル0の間にテクスチャフィルタから出力されるテクセルが色変換されるべきかどうか

かを特定する、テクセル変換モードフィールド、を解釈する回路と、

(W) クロマキーボード入力を選択的に可能にする、クロマキーイネーブルモードフィールド、を解釈する回路と、

(V2) RGBディザリングのタイプを選択する、RGBディザ選択モードフィールド、を解釈する回路と、

(V1) アルファディザリングのタイプを選択する、アルファディザ選択モードフィールド、を解釈する回路と、

(V) ブレンドパラメータを特定する、複数のブレンドモードワード、を解釈する回路と、

(M) 前記ブレンドが強制使用可能にされるべきかどうかを特定する、強制ブレンドイネーブルモードフィールド、を解釈する回路と、

(L) 有効範囲が画素アルファを決定するために用いられるべきかどうかを特定する、アルファ有効範囲選択モードフィールド、を解釈する回路と、

(K) アルファによって増大された有効範囲が画素アルファおよび有効範囲を決定するために用いられるべきかどうかを特定する、有効範囲タイムスアルファ選択モードフィールド、を解釈する回路と、

(J) Zバッファリングを特定する、Zモード選択モードフィールド、を解釈する回路と、

(I) 有効範囲宛先を特定する、有効範囲宛先モードフィールド、を解釈する回路と、

(H) 色が有効範囲からはみ出したところでのみ更新されるべきかどうかを特定する、有効範囲のカラーモードフィールド、を解釈する回路と、

(G) 色および/または有効範囲読み出し/修正/書き込みフレームバッファメモリアクセスを選択的に可能にする、画像読み出しイネーブルモードフィールド、を解釈する回路と、

(F) 色書き込みが可能にされるかどうかによってZバッファ書き込みを選択的に可能にする、Zアップデートイネーブルモードフィールド、を解釈する回路と、

(E) 奥行比較において可能な条件付色書き込みを特定する、Z比較イネーブルモードフィールド、を解釈する回路と、

(D) 有効範囲を用いたブレンドイネーブルを許可する、アンチエリアスイネーブルモードフィールド、を解釈する回路と、

(C) プリミティブ奥行と画素奥行の間で選択をする、Zソース選択モードフィールド、を解釈する回路と、

(B) ランダムノイズがアルファ比較に用いられるべきかどうかを特定する、ディザアルファイネーブルモードフィールド、を解釈する回路と、

(A) アルファ比較上で条件付色書き込みを可能とする、アルファ比較イネーブルモードフィールド、を解釈する回路と、

画像を生成するために、上記の回路(a)～(k)、
(A)～(M)、(V1)、(V2)、(W)～(Z)
に結合された回路と、
を含む、システム。

【請求項32】 画素毎の1サイクルモード、画素毎の
2サイクルモード、コピーモード、およびフィルモード
の表示パイプラインサイクル制御モードを選択するサイ
クルタイプモードを解釈する手段を含む、請求項31に
記載のシステム。

【請求項33】 (1) 5ビットの赤、5ビットの緑、
5ビットの青、および1ビットのアルファのRGBAフ
ォーマット内のカラーlookupアップテーブル中における
テクセルの記憶、

(2) 8ビット輝度値および8ビットアルファ値を供給
する輝度アルファフォーマット内のカラーlookupア
ップテーブル中におけるテクセルの記憶、
のうちのいずれかを選択するテクスチュアルlookupア
ップモードフィールドを解釈する手段を含む、請求項3
1に記載のシステム。

【請求項34】 (1) ポイントサンプリング、

(2) 2×2配列サンプリング

のうちのいずれかを選択するサンブルタイプモードフ
ィールドを解釈する手段を含む、請求項31に記載のシ
ステム。

【請求項35】 (1) マジックスクエアマトリッ
クス、

(2) ベイヤーマトリックス、

(3) ノイズ、

(4) ディザリング無し、

に基づくディザリングのうちのいずれかを選択するRG
Bディザ選択モードフィールドを解釈する手段を含む、
請求項31に記載のシステム。

【請求項36】 (1) 所定のパターン、

(2) 所定のパターンのネガティブ、

(3) ノイズ、

(4) ディザリング無し、

に基づくディザリングを特定するアルファディザリン
グ選択モードフィールドを解釈する手段を含む、請求項3
1に記載のシステム。

【請求項37】 パイプラインサイクル0の間、第1の
ブレンダ入力を選択的な増大、

パイプラインサイクル1の間、第1のブレンダ入力
を選択的な増大、

パイプラインサイクル0の間、第2のブレンダ入力
を選択的な増大、

パイプラインサイクル1の間、第2のブレンダ入力
を選択的な増大、

パイプラインサイクル0の間、第3のブレンダ入力
を選択的な増大、

パイプラインサイクル1の間、第3のブレンダ入力

選択的な増大、

パイプラインサイクル0の間、第4のブレンダ入力
を選択的な増大、

パイプラインサイクル1の間、第4のブレンダ入力
を選択的な増大、

を特定するブレンダパラメータを特定する複数のブ
レンドモードワードを解釈する手段を含む、請求項31
に記載のシステム。

【請求項38】 (1) クランプ、

(2) ラップ、

(3) 全有効範囲への作用、

(4) 保存、

のうちのいずれかの有効範囲宛先モードを選択する有
効範囲宛先モードフィールドを解釈する手段を含む、請
求項31に記載のシステム。

【請求項39】 (1) 不透明、

(2) 浸透、

(3) 透明、

(4) デカル、

のうちのいずれかのZバッファリングモードを選択す
るZモード選択モードフィールドを解釈する手段を含む、
請求項31に記載のシステム。

【請求項40】 少なくとも1つの3次元表示モード制
御コマンドを記憶し、3次元グラフィックシステムで使
用するための記憶媒体であって、当該3次元表示モード
制御コマンドは、

101111の6ビットバイナリ値を含むコマンド識別
子フィールドと、

下記のモード制御フィールド、つまり、

(k) 次のプリミティブを読み取る前にフレームバッ
ファにプリミティブを書き込ませるかどうかを特定する、
アトミックプリミティブモードフィールド、

(i) 表示パイプラインサイクル制御モードを選択す
る、サイクルタイプモードフィールド、

(h) 遠近法テクスチャ修正を選択的に可能にする、
遠近法テクスチャイネーブルモードフィールド、

(g) テクスチャの詳細な処理を選択的に可能にす
る、テクスチャ詳細モードフィールド、

(f) テクスチャを鮮明にすることを選択的に可能に
する、テクスチャ鮮明イネーブルモード、

(e) テクスチャの詳細レベル値の処理を選択的に可
能にする、テクスチャ詳細イネーブルモード、

(d) カラーlookupアップテーブルからテクスチャ値
を選択的に検索することを可能にする、イネーブルル
ックアップテーブルモードフィールド、

(c) カラーlookupアップテーブル中のテクセルのタイ
プを特定する、テクスチュアルlookupアップテーブル
タイプモードフィールド、

(b) テクセルがどのようにサンプリングされるべきか
を特定する、サンブルタイプモードフィールド、

(a) テクセルが2×2ハーフテクセル補間を用いてフィルタリングされるべきかどうかを特定する、ミッドテクセルモードフィールド、

(Z) テクスチャフィルタがバイブラインサイクル0中のテクセルをバイリニア的に補間すべきかどうかを特定する、第1のバイラップモードフィールド、

(Y) テクスチャフィルタがバイブラインサイクル1中のテクセルをバイリニア的に補間すべきかどうかを特定する、第2のバイラップモードフィールド、

(X) バイブラインサイクル0の間にテクスチャフィルタから出力されるテクセルが色変換されるべきかどうかを特定する、テクセル変換モードフィールド、

(W) クロマキーボード入力を選択的に可能にする、クロマキーイネーブルモードフィールド、

(V2) RGBディザリングのタイプを選択する、RGBディザ選択モードフィールド、

(V1) アルファディザリングのタイプを選択する、アルファディザ選択モードフィールド、

(V) ブレンダパラメータを特定する、複数のブレンダーモードワード、

(M) 前記ブレンダが強制使用可能にされるべきかどうかを特定する、強制ブレンドイネーブルモードフィールド、

(L) 有効範囲が画素アルファを決定するために用いられるべきかどうかを特定する、アルファ有効範囲選択モードフィールド、

(K) アルファによって増大された有効範囲が画素アルファおよび有効範囲を決定するために用いられるべきかどうかを特定する、有効範囲タイムスアルファ選択モードフィールド、

(J) Zバッファリングを特定する、Zモード選択モードフィールド、

(I) 有効範囲宛先を特定する、有効範囲宛先モードフィールド、

(H) 色が有効範囲からはみ出したところでのみ更新されるべきかどうかを特定する、有効範囲のカラーモードフィールド、

(G) 色および/または有効範囲読み出し/修正/書き込みフレームバッファメモリアクセスを選択的に可能にする、画像読み出しイネーブルモードフィールド、

(F) 色書き込みが可能にされるかどうかによってZバッファ書き込みを選択的に可能にする、Zアップデートイネーブルモードフィールド、

(E) 奥行比較において可能な条件付色書き込みを特定する、Z比較イネーブルモードフィールド、

(D) 有効範囲を用いたブレンドイネーブルを許可する、アンチエリアスイネーブルモードフィールド、

(C) プリミティブ奥行と画素奥行の間で選択をする、Zソース選択モードフィールド、

(B) ランダムノイズがアルファ比較に用いられるべき

かどうかを特定する、ディザアルファイネーブルモードフィールド、

(A) アルファ比較上で条件付色書き込みを可能とする、アルファ比較イネーブルモードフィールド

のうちの少なくとも1つを含む、記憶媒体。

【請求項41】 画素毎の1サイクルモード、画素毎の2サイクルモード、コピーモード、およびフィルモードの表示バイブラインサイクル制御モードを選択するサイクルタイプモードを記憶する手段を含む、請求項40に記載の記憶媒体。

【請求項42】 (1) 5ビットの赤、5ビットの緑、5ビットの青、および1ビットのアルファのRGBAフォーマット内のカラールックアップテーブル中におけるテクセルの記憶、

(2) 8ビット輝度値および8ビットアルファ値を供給する輝度アルファフォーマット内のカラールックアップテーブル中におけるテクセルの記憶、

のうちのいずれかを選択するテクスチャルックアップタイプモードフィールドを記憶する手段を含む、請求項40に記載の記憶媒体。

【請求項43】 (1) ポイントサンプリング、

(2) 2×2配列サンプリング、

のうちのいずれかを選択をするサンブルタイプモードフィールドを記憶する手段を含む、請求項40に記載の記憶媒体。

【請求項44】 (1) マジックスクエアマトリックス、

(2) ベイヤーマトリックス、

(3) ノイズ、

(4) ディザリング無し、

に基づくディザリングのうちのいずれかを選択するRGBディザ選択モードフィールドを記憶する手段を含む、請求項40に記載の記憶媒体。

【請求項45】 (1) 所定のパターン、

(2) 所定のパターンのネガティブ、

(3) ノイズ、

(4) ディザリング無し、

に基づくディザリングを特定するアルファディザリング選択モードフィールドを記憶する手段を含む、請求項40に記載の記憶媒体。

【請求項46】 バイブラインサイクル0の間、第1のブレンダ入力の選択的な増大、

バイブラインサイクル1の間、第1のブレンダ入力の選択的な増大、

バイブラインサイクル0の間、第2のブレンダ入力の選択的な増大、

バイブラインサイクル1の間、第2のブレンダ入力の選択的な増大、

バイブラインサイクル0の間、第3のブレンダ入力の選択的な増大、

バイブラインサイクル1の間、第3のブレンダ入力の選択的な増大、
 バイブラインサイクル0の間、第4のブレンダ入力の選択的な増大、
 バイブラインサイクル1の間、第4のブレンダ入力の選択的な増大、
 を特定するブレンダパラメータを特定する複数のブレンドモードワードを記憶する手段を含む、請求項40に記載の記憶媒体。

- 【請求項47】 (1) クランプ、
 (2) ラップ、
 (3) 全有効範囲への作用、
 (4) 保存、

のうちのいずれかの有効範囲宛先モードを選択する有効範囲宛先モードフィールドを記憶する手段を含む、請求項40に記載の記憶媒体。

- 【請求項48】 (1) 不透明、
 (2) 浸透、
 (3) 透明、
 (4) デカル、

のうちのいずれかのZバッファリングモードを選択するZモード選択モードフィールドを記憶する手段を含む、請求項40に記載の記憶媒体。

【請求項49】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示モード制御コマンドを生成するプロセスであって、少なくとも1つのセットモードコマンドを生成するステップを含み、当該セットモードコマンドは、

111100の6ビットバイナリ値を含むコマンド識別子フィールドと、

下記の付加フィールド、つまり、

色複合器から少なくとも1つの色スペース値を減算することを特定する、少なくとも1つのコンバイナ減算モード制御フィールド、

色複合器入力に少なくとも1つの色スペース値を乗算することを特定する、少なくとも1つのコンバイナ乗算モード制御フィールド、

カラーコンバイナ加算回路入力に特定する、少なくとも1つのコンバイナ加算制御フィールド、

のうちの少なくとも1つを有する、プロセス。

【請求項50】 マルチプレクサソースを指定し、関数 $(A-B) * C + D$ を実現するために、以下の複合器減算モード制御フィールド、つまり、

- (1) 減算ソースAを特定する減算ソース制御フィールドと、
 (2) 減算ソースBを特定する減算ソース制御フィールドと、
 (3) 乗算ソースCを特定する乗算ソース制御フィールドと、
 (4) 加算ソースDを特定する加算ソース制御フィールドと、

ドと、

を生成するステップを含む、請求項49に記載のプロセス。

【請求項51】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

- (1) RGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
 (2) アルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
 (3) RGB構成要素減算ソースBを特定する減算ソース制御フィールドと、
 (4) アルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、
 (5) RGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、
 (6) アルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、
 (7) RGB構成要素加算ソースDを特定する加算ソース制御フィールドと、
 (8) アルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

を生成するステップをさらに含む、請求項49に記載のプロセス。

【請求項52】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

- (1) バイブラインサイクル0のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
 (2) バイブラインサイクル1のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
 (3) バイブラインサイクル0のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
 (4) バイブラインサイクル1のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
 (5) バイブラインサイクル0のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、
 (6) バイブラインサイクル1のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、
 (7) バイブラインサイクル0のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、
 (8) バイブラインサイクル1のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、
 (9) バイブラインサイクル0のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、
 (10) バイブラインサイクル0のためのアルファ構成

要素乗算ソースCを特定する乗算ソース制御フィールドと、

(11) バイブラインサイクル1のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(12) バイブラインサイクル1のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(13) バイブラインサイクル0のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(14) バイブラインサイクル0のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

(15) バイブラインサイクル1のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(16) バイブラインサイクル1のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

を生成するステップをさらに含む、請求項49に記載のプロセス。

【請求項53】 RGBカラーコンバイナチャンネルに対応する入力を選択する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値の生成と、

アルファカラーコンバイナチャンネルに対応する入力を選択する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値の生成と、をさらに含む、請求項49に記載のプロセス。

【請求項54】 バイブラインサイクル0のカラーコンバイン操作に対応する入力を選択する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値の生成と、

バイブラインサイクル1のカラーコンバイン操作に対応する入力を選択する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値の生成と、をさらに含む、請求項49に記載のプロセス。

【請求項55】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示モード制御コマンドを生成するシステムであって、

少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

少なくとも1つのセットモードコマンドを供給するために、前記プロセッサと前記メモリに結合される回路と、

を含み、前記セットモードコマンドは、

111100の6ビットバイナリ値を含むコマンド識別子フィールドと、

以下の付加フィールド、つまり、

カラーコンバイナからの少なくとも1つのカラースペース値の減算を特定する少なくとも1つのコンバイナ減算

モード制御フィールド、

少なくとも1つのカラースペース値によるカラーコンバイナ入力の乗算を特定する少なくとも1つのコンバイナ乗算モード制御フィールド、

カラーコンバイナ加算入力を特定する少なくとも1つのコンバイナ加算制御フィールド、のなかの少なくとも1つを有する、システム。

【請求項56】 $(A - B) * C + D$ の関数を実現するマルチプレクサソースを特定するための以下のコンバイナ減算モード制御フィールド、つまり、

(1) 減算ソースAを特定する減算ソース制御フィールドと、

(2) 減算ソースBを特定する減算ソース制御フィールドと、

(3) 乗算ソースCを特定する乗算ソース制御フィールドと、

(4) 加算ソースDを特定する加算ソース制御フィールドと、

を供給するための手段をさらに含む、請求項55に記載のシステム。

【請求項57】 $(A - B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

(1) RGB構成要素減算ソースAを特定する減算ソース制御フィールドと、

(2) アルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、

(3) RGB構成要素減算ソースBを特定する減算ソース制御フィールドと、

(4) アルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、

(5) RGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(6) アルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(7) RGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(8) アルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

を生成するための手段をさらに含む、請求項55に記載のシステム。

【請求項58】 $(A - B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

(1) バイブラインサイクル0のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、

(2) バイブラインサイクル1のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、

(3) バイブラインサイクル0のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールド

10

20

30

40

50

と、

(4) バイブラインサイクル1のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、

(5) バイブラインサイクル0のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、

(6) バイブラインサイクル1のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、

(7) バイブラインサイクル0のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、

(8) バイブラインサイクル1のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、

(9) バイブラインサイクル0のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(10) バイブラインサイクル0のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(11) バイブラインサイクル1のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(12) バイブラインサイクル1のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(13) バイブラインサイクル0のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(14) バイブラインサイクル0のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

(15) バイブラインサイクル1のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(16) バイブラインサイクル1のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、
を生成する回路をさらに含む、請求項55に記載のシステム。

【請求項59】 RGBカラーコンバイナチャンネルに対応する入力を選択する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値を供給する回路と、アルファカラーコンバイナチャンネルに対応する入力を選択する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値を供給する手段と、をさらに含む、請求項55に記載のシステム。

【請求項60】 バイブラインサイクル0のカラーコンバイナ操作に対応する入力を選択する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値を生成する回路と、

バイブラインサイクル1のカラーコンバイナ操作に対応する入力を選択する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値を生成する回路と、をさらに含む、請求項55に記載のシステム。

【請求項61】 3次元グラフィックス表示システムにおいて、少なくとも1つのセットモードコマンドを解釈するプロセスであって、当該セットモードコマンドは、
(a) 111100の6ビットバイナリ値を含むコマンド識別子フィールドの解釈と、

(b) 以下の付加フィールド、つまり、
カラーコンバイナからの少なくとも1つのカラースペース値の減算を特定する、少なくとも1つのコンバイナ減算モード制御フィールド、
カラーコンバイナ入力と少なくとも1つのカラースペース値との乗算を特定する、少なくとも1つのコンバイナ乗算モード制御フィールド、
カラーコンバイナ加算入力を特定する、少なくとも1つのコンバイナ加算制御フィールド、のなかの少なくとも1つの解釈と、
(c) 少なくとも部分的にステップ(b)に基づく画像の生成と、
を備える、プロセス。

【請求項62】 $(A - B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

(1) 減算ソースAを特定する減算ソース制御フィールドと、

(2) 減算ソースBを特定する減算ソース制御フィールドと、

(3) 乗算ソースCを特定する乗算ソース制御フィールドと、

(4) 加算ソースDを特定する加算ソース制御フィールドと、

を解釈するステップをさらに含む、請求項61に記載のプロセス。

【請求項63】 $(A - B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

(1) RGB構成要素減算ソースAを特定する減算ソース制御フィールドと、

(2) アルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、

(3) RGB構成要素減算ソースBを特定する減算ソース制御フィールドと、

(4) アルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、

(5) RGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(6) アルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(7) RGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(8) アルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

を解釈するステップをさらに含む、請求項61に記載のプロセス。

【請求項64】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

(1) バイブラインサイクル0のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、

(2) バイブラインサイクル1のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、

(3) バイブラインサイクル0のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、

(4) バイブラインサイクル1のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、

(5) バイブラインサイクル0のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、

(6) バイブラインサイクル1のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、

(7) バイブラインサイクル0のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、

(8) バイブラインサイクル1のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、

(9) バイブラインサイクル0のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(10) バイブラインサイクル0のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(11) バイブラインサイクル1のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(12) バイブラインサイクル1のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(13) バイブラインサイクル0のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(14) バイブラインサイクル0のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

(15) バイブラインサイクル1のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(16) バイブラインサイクル1のためのアルファ構成

要素加算ソースDを特定する加算ソース制御フィールドと、

を解釈するステップをさらに含む、請求項61に記載のプロセス。

【請求項65】 RGBカラーコンバイナチャンネルに対応する入力を特定する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値の解釈と、

アルファカラーコンバイナチャンネルに対応する入力を特定する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値の解釈と、をさらに含む、請求項61に記載のプロセス。

【請求項66】 バイブラインサイクル0のカラーコンバイン操作に対応する入力を特定する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値の解釈と、

バイブラインサイクル1のカラーコンバイン操作に対応する入力を特定する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値の解釈と、をさらに含む、請求項61に記載のプロセス。

【請求項67】 111100の6ビットバイナリ値を含む、少なくとも1つのセットモードコマンドを解釈するための3次元グラフィックス表示システムであって、

(a) 111100の6ビットバイナリ値を含むコマンド識別子フィールドを解釈する手段と、

(b) 以下の付加フィールド、つまり、カラーコンバイナからの少なくとも1つのカラースペース値の減算を特定する少なくとも1つのコンバイナ減算モード制御フィールド、

カラーコンバイナ入力と少なくとも1つのカラースペース値との乗算を特定する少なくとも1つのコンバイナ乗算モード制御フィールド、

カラーコンバイナ加算入力を特定する少なくとも1つのコンバイナ加算制御フィールド、

のうちの少なくとも1つに基づいて、少なくとも1つの制御信号を生成する手段と、

(c) 少なくとも部分的に前記制御信号に基づいて、画像を生成する手段と、を備える、システム。

【請求項68】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

(1) 減算ソースAを特定する減算ソース制御フィールドと、

(2) 減算ソースBを特定する減算ソース制御フィールドと、

(3) 乗算ソースCを特定する乗算ソース制御フィールドと、

(4) 加算ソースDを特定する加算ソース制御フィールドと、

を解釈するコンバイナ回路をさらに含む、請求項67に

記載のシステム。

【請求項69】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

- (1) RGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
 - (2) アルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
 - (3) RGB構成要素減算ソースBを特定する減算ソース制御フィールドと、
 - (4) アルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、
 - (5) RGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、
 - (6) アルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、
 - (7) RGB構成要素加算ソースDを特定する加算ソース制御フィールドと、
 - (8) アルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、
- を解釈するコンバイナをさらに含む、請求項67に記載のシステム。

【請求項70】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

- (1) バイブラインサイクル0のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (2) バイブラインサイクル1のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (3) バイブラインサイクル0のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (4) バイブラインサイクル1のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (5) バイブラインサイクル0のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、
- (6) バイブラインサイクル1のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、
- (7) バイブラインサイクル0のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、
- (8) バイブラインサイクル1のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、
- (9) バイブラインサイクル0のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、
- (10) バイブラインサイクル0のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(11) バイブラインサイクル1のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(12) バイブラインサイクル1のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(13) バイブラインサイクル0のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

10 (14) バイブラインサイクル0のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

(15) バイブラインサイクル1のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(16) バイブラインサイクル1のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

20 に基づいた色信号を結合するコンバイナをさらに含む、請求項67に記載のシステム。

【請求項71】 RGBカラーコンバイナチャンネルに対応する入力を特定する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値を解釈する手段と、アルファカラーコンバイナチャンネルに対応する入力を特定する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値を解釈する手段と、をさらに含む、請求項67に記載のシステム。

30 【請求項72】 バイブラインサイクル0のカラーコンバイナ操作に対応する入力を特定する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値を解釈する手段と、

バイブラインサイクル1のカラーコンバイナ操作に対応する入力を特定する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値を解釈する手段と、をさらに含む、請求項67に記載のシステム。

【請求項73】 3次元グラフィックスシステムに使用する記憶媒体であって、当該記憶媒体は少なくとも1つの表示モード制御コマンドを記憶し、当該表示モード制御コマンドは、

40 111100の6ビットバイナリ値を含むコマンド識別子フィールドと、

以下の付加フィールド、つまり、

カラーコンバイナからの少なくとも1つのカラースペース値の減算を特定する、少なくとも1つのコンバイナ減算モード制御フィールド、

少なくとも1つのカラースペース値によるカラーコンバイナ入力の乗算を特定する、少なくとも1つのコンバイナ乗算モード制御フィールド、

50 カラーコンバイナ加算入力を特定する、少なくとも1つのコンバイナ加算制御フィールド、の少なくとも1つを

有する、記憶媒体。

【請求項74】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

- (1) 減算ソースAを特定する減算ソース制御フィールドと、
- (2) 減算ソースBを特定する減算ソース制御フィールドと、
- (3) 乗算ソースCを特定する乗算ソース制御フィールドと、
- (4) 加算ソースDを特定する加算ソース制御フィールドと、

を記憶する手段をさらに含む、請求項73に記載の記憶媒体。

【請求項75】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

- (1) RGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (2) アルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (3) RGB構成要素減算ソースBを特定する減算ソース制御フィールドと、
- (4) アルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、
- (5) RGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、
- (6) アルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、
- (7) RGB構成要素加算ソースDを特定する加算ソース制御フィールドと、
- (8) アルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

を記憶する手段をさらに含む、請求項73に記載の記憶媒体。

【請求項76】 $(A-B) * C + D$ の関数を実現するマルチプレクサソースを特定するための、以下のコンバイナ減算モード制御フィールド、つまり、

- (1) バイブラインサイクル0のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (2) バイブラインサイクル1のためのRGB構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (3) バイブラインサイクル0のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (4) バイブラインサイクル1のためのアルファ構成要素減算ソースAを特定する減算ソース制御フィールドと、
- (5) バイブラインサイクル0のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、

を記憶する手段をさらに含む、請求項73に記載の記憶媒体。

(6) バイブラインサイクル1のためのRGB構成要素減算ソースBを特定する減算ソース制御フィールドと、

(7) バイブラインサイクル0のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、

(8) バイブラインサイクル1のためのアルファ構成要素減算ソースBを特定する減算ソース制御フィールドと、

(9) バイブラインサイクル0のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(10) バイブラインサイクル0のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(11) バイブラインサイクル1のためのRGB構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(12) バイブラインサイクル1のためのアルファ構成要素乗算ソースCを特定する乗算ソース制御フィールドと、

(13) バイブラインサイクル0のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(14) バイブラインサイクル0のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

(15) バイブラインサイクル1のためのRGB構成要素加算ソースDを特定する加算ソース制御フィールドと、

(16) バイブラインサイクル1のためのアルファ構成要素加算ソースDを特定する加算ソース制御フィールドと、

を記憶する手段をさらに含む、請求項73に記載の記憶媒体。

【請求項77】 RGBカラーコンバイナチャンネルに対応する入力を特定する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値を記憶する手段と、アルファカラーコンバイナチャンネルに対応する入力を特定する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値を記憶する手段と、をさらに含む、請求項73に記載の記憶媒体。

【請求項78】 バイブラインサイクル0のカラーコンバイン操作と対応する入力を特定する、少なくとも第1、第2、第3および第4のマルチプレクサ選択値を記憶する手段と、

バイブラインサイクル1のカラーコンバイン操作と対応する入力を特定する、少なくとも第5、第6、第7および第8のマルチプレクサ選択値を記憶する手段と、をさらに含む、請求項73に記載の記憶媒体。

【請求項79】 3次元グラフィックスシステムによって処理するための、少なくとも1つのカラー画像モード

コマンドを生成するプロセスであって、当該プロセスは、少なくとも1つのコマンドを生成するステップを含み、当該コマンドは、

111111と111101との設定内の6ビットバイナリ値を含むコマンド識別子フィールドと、
画像データ初期化パラメータと、
色構成要素範囲パラメータと、
画像幅パラメータと、
ベースアドレスパラメータと、を含む、プロセス。

【請求項80】 (a) r g b a、
(b) y u v、
(c) カラーインデックス、
(d) 強度アルファ、
(e) アルファ、

のいずれかを選択する画像データ初期化パラメータを生成するステップをさらに含む、請求項79に記載のプロセス。

【請求項81】 (a) 4ビット幅色構成要素値、
(b) 8ビット幅色構成要素値、
(c) 16ビット幅色構成要素値、
(d) 32ビット幅色構成要素値、

のいずれかを選択する色構成要素範囲パラメータを生成するステップをさらに含む、請求項79に記載のプロセス。

【請求項82】 メモリに記憶された画像の画素の幅を特定する画像幅パラメータ値を生成するステップをさらに含む、請求項79に記載のプロセス。

【請求項83】 画像の上部左端のメインメモリにおけるベースアドレスを特定するベースアドレスパラメータを生成するステップをさらに含む、請求項79に記載のプロセス。

【請求項84】 3次元グラフィックスシステムによって処理するための少なくとも1つのカラー画像モードコマンドを備えるシステムであって、当該システムは、少なくとも1つのプロセッサと、
少なくとも1つのメモリと、
少なくとも1つのコマンドを生成するために前記プロセッサと前記メモリに結合される回路と、を含み、前記コマンドは、

111111と111101との設定内で6ビットバイナリ値を含むコマンド識別子フィールドと、
画像データ初期化パラメータと、
色構成要素範囲パラメータと、
画像幅パラメータと、
ベースアドレスパラメータと、を含む、システム。

【請求項85】 (a) r g b a、
(b) y u v、
(c) カラーインデックス、
(d) 強度アルファ、
(e) アルファ、

のいずれかを選択する画像データ初期化パラメータを供給する手段をさらに含む、請求項84に記載のシステム。

【請求項86】 (a) 4ビット幅色構成要素値、
(b) 8ビット幅色構成要素値、
(c) 16ビット幅色構成要素値、
(d) 32ビット幅色構成要素値、

のいずれかを選択する色構成要素範囲パラメータを備える手段をさらに含む、請求項84に記載のシステム。

10 【請求項87】 メモリに記憶された画像の画素の幅を特定する画像幅パラメータ値を備える手段をさらに含む、請求項84に記載のシステム。

【請求項88】 画像の上部左端のメインメモリにおけるベースアドレスを特定するベースアドレスパラメータを備える手段をさらに含む、請求項84に記載のシステム。

【請求項89】 3次元グラフィックスシステムにおいて、少なくとも1つのカラー画像モードコマンドを実行するプロセスであって、当該カラー画像モードコマンドは、

20 (a) カラー画像モードに対応する111111と111101の設定内で、6ビットバイナリ値を含むコマンド識別子フィールドの解釈と、

(b) 画像データ初期化パラメータ、
色構成要素範囲パラメータ、
画像幅パラメータ、
ベースアドレスパラメータ、の少なくとも1つの解釈と、

30 (c) 少なくとも部分的にステップ(b)に基づいた、色画像表示の生成と、
を含む、プロセス。

【請求項90】 画像データ初期化パラメータに基づいて、

(a) r g b a、
(b) y u v、
(c) カラーインデックス、
(d) 強度アルファ、
(e) アルファ、

のいずれかを選択するステップをさらに含む、請求項89に記載のプロセス。

【請求項91】 画像データ初期化パラメータに基づいて、

(a) 4ビット幅色構成要素値、
(b) 8ビット幅色構成要素値、
(c) 16ビット幅色構成要素値、
(d) 32ビット幅色構成要素値、

のいずれかを選択するステップをさらに含む、請求項89に記載のプロセス。

50 【請求項92】 画像幅パラメータ値に基づいて、メモリに記憶された画像の画素の幅を設定するステップをさ

らに含む、請求項89に記載のプロセス。

【請求項93】 ベースアドレスパラメータに基づいて、画像の上部左端のメインメモリにおけるベースアドレスを設定するステップをさらに含む、請求項89に記載のプロセス。

【請求項94】 カラー画像モードに対応する111111と111101の設定内で、6ビットバイナリ値を含むコマンド識別子を有する、少なくとも1つのカラー画像モードコマンドを実行する3次元グラフィックスシステムであって、当該システムは、カラー画像モードに対応する111111と111101との設定内の6ビットバイナリ値を含むコマンド識別子を解釈するためのコマンド識別子復号器と、画像データ初期化パラメータ、色構成要素範囲パラメータ、画像幅パラメータ、ベースアドレスパラメータ、の少なくとも1つを解釈するためのコマンドパラメータ復号器と、少なくとも部分的にパラメータに基づく色画像表示を生成する表示回路と、を含む、システム。

【請求項95】 画像データ初期化パラメータに基づいて、

- (a) r g b a、
- (b) y u v、
- (c) カラーインデックス、
- (d) 強度アルファ、
- (e) アルファ、

のいずれかを選択する構成をさらに含む、請求項94に記載のシステム。

【請求項96】 画像データ初期化パラメータに基づいて、

- (a) 4ビット幅色構成要素値、
- (b) 8ビット幅色構成要素値、
- (c) 16ビット幅色構成要素値、
- (d) 32ビット幅色構成要素値、

のいずれかを選択する構成をさらに含む、請求項94に記載のシステム。

【請求項97】 画像幅パラメータ値に基づいて、メモリに記憶された画像の画素の幅を設定する手段をさらに含む、請求項94に記載のシステム。

【請求項98】 ベースアドレスパラメータに基づいて、画像の上部左端のメインメモリにおけるベースアドレスを設定する回路をさらに含む、請求項94に記載のシステム。

【請求項99】 少なくとも1つのカラー画像モードコマンドを記憶し、3次元グラフィックスシステムに使用する記憶媒体であって、当該カラー画像モードコマンドは、111111と111101との設定内で6ビットバイナリ値を含むコマンド識別子フィールドと、

画像データ初期化パラメータと、色構成要素範囲パラメータと、画像幅パラメータと、ベースアドレスパラメータと、を含む、記憶媒体。

- 【請求項100】 (a) r g b a、
(b) y u v、
(c) カラーインデックス、
(d) 強度アルファ、
(e) アルファ、

10 のいずれかを選択する画像データ初期化パラメータを記憶する手段をさらに含む、請求項99に記載の記憶媒体。

- 【請求項101】 (a) 4ビット幅色構成要素値、
(b) 8ビット幅色構成要素値、
(c) 16ビット幅色構成要素値、
(d) 32ビット幅色構成要素値、

のいずれかを選択する色構成要素範囲パラメータを記憶する手段をさらに含む、請求項99に記載の記憶媒体。

【請求項102】 メモリに記憶された画像の画素の幅を設定する画像幅パラメータ値を記憶する手段をさらに含む、請求項99に記載の記憶媒体。

【請求項103】 画像の上部左端のメインメモリにおけるベースアドレスを特定するベースアドレスパラメータを記憶する手段をさらに含む、請求項99に記載の記憶媒体。

【請求項104】 3次元グラフィックスシステムによって処理するための、少なくとも1つのマスク画像モードコマンドを生成するプロセスであって、少なくとも1つの設定マスク画像コマンドを生成するステップを含み、当該設定マスク画像コマンドは、111110の6ビットバイナリ値を含むコマンド識別子フィールドと、少なくとも1つの奥行画像の上部左端のメモリアドレスを特定するベースアドレスと、を含む、プロセス。

【請求項105】 3次元グラフィックスシステムによって処理するための、少なくとも1つのマスク画像モードコマンドを備えるシステムであって、少なくとも1つのプロセッサと、

40 少なくとも1つのメモリと、少なくとも1つの設定マスク画像コマンドを備える前記プロセッサと前記メモリとに結合された手段と、を含み、前記設定マスク画像コマンドは、111110の6ビットバイナリ値を含むコマンド識別子フィールドと、少なくとも1つの奥行画像の上部左端の記憶アドレスを特定するベースアドレスと、を含む、システム。

【請求項106】 3次元グラフィックスシステムにおいて、少なくとも1つのマスク画像モードコマンドを解釈するプロセスであって、当該マスク画像モードコマン

ドは、
111110の6ビットバイナリ値を含むコマンド識別子フィールドの解釈と、
少なくとも1つの奥行画像の上部左端のメモリアドレスを特定するベースアドレスの解釈と、を含む、プロセス。

【請求項107】 3次元グラフィックスシステムにおいて、111110の6ビットバイナリ値を含む、少なくとも1つのマスク画像モードコマンドを解釈する復号器であって、当該システムは、
111110の6ビットバイナリ値を含むコマンド識別子フィールドを解釈する手段と、
少なくとも1つの奥行画像の上部左端のメモリアドレスを特定するベースアドレスを解釈する手段と、を含む、システム。

【請求項108】 少なくとも1つのマスク画像モードコマンドを記憶し、3次元グラフィックスシステムに使用する記憶媒体であって、当該マスク画像モードコマンドは、
111110の6ビットバイナリ値を含むコマンド識別子フィールドと、
少なくとも1つの奥行画像の上部左端のメモリアドレスを特定するベースアドレスと、を含む、記憶媒体。

【請求項109】 3次元グラフィックスシステムで処理するための、少なくとも1つの表示コマンドを生成するプロセスであって、少なくともxおよびy位置値で、001000から001111の範囲内に6ビットバイナリ値を含むコマンド識別子フィールドを有する少なくとも1つの三角描画コマンドと、xおよびy位置値の対応するxおよびy位置で、少なくとも1つの三角を特定する三角描画コマンドフォーマットと、を生成するステップを含むプロセス。

【請求項110】 3次元グラフィックスシステムで処理するための、少なくとも1つの表示コマンドを生成するシステムであって、
少なくとも1つのプロセッサと、
少なくとも1つのメモリと、
少なくともxおよびy位置値で、001000から001111の範囲内に6ビットバイナリ値を含むコマンド識別子フィールドを有する少なくとも1つの三角描画コマンドと、xおよびy位置値の対応するxおよびy位置で、少なくとも1つの三角を特定する三角描画コマンドフォーマットと、を供給するプロセッサとメモリを結合した手段と、を含む、システム。

【請求項111】 3次元グラフィックスシステムにおいて、少なくとも1つの表示コマンドを実行し、処理するプロセスであって、

(a) 少なくともxおよびy位置値が、001000から001111の範囲内の6ビットバイナリ値を含むコマンド識別子フィールドを有する少なくとも1つの三角

描画コマンドを解釈するステップと、

(b) 前記xおよびy位置値に対応するxおよびy位置において、少なくとも1つのプリミティブをレンダリングするステップと、を含む、プロセス。

【請求項112】 少なくともxおよびy位置値が、001000から001111の範囲内の6ビットバイナリ値を含む、少なくとも1つの表示コマンドを実行する3次元グラフィックスシステムであって、
少なくともxおよびy位置値が、001000から001111の範囲内の6ビットバイナリ値を含むコマンド識別子フィールドを有する少なくとも1つの三角描画コマンドを解釈する復号器と、
前記xおよびy位置値に対応するxおよびy位置において、少なくとも1つのプリミティブをレンダリングする表示プロセッサと、を含む、システム。

【請求項113】 3次元グラフィックスシステムで用いられる記憶媒体であって、少なくともxおよびy位置値が、001000から001111の範囲内の6ビットバイナリ値を含むコマンド識別子フィールドを有する少なくとも1つの表示コマンドと、当該xおよびy位置値に対応するxおよびy位置において、少なくとも1つの三角を特定する三角描画コマンドフォーマットと、を記憶する記憶媒体。

【請求項114】 少なくとも1つの表示コマンドを生成する3次元グラフィックスシステムで処理するためのプロセスであって、少なくとも1つの三角描画コマンドを生成するステップを含み、当該三角描画コマンドは、001111と001011のうちの少なくとも1つにおいて、6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットのエッジ係数と、

1セットのテクスチャ係数と、

1セットのZバッファ係数と、

を含み、前記三角描画コマンドフォーマットは、少なくとも1つの三角が前記1セットのエッジ係数にしたがって描画され、少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされ、少なくとも部分的に前記1セットのバッファ係数に基づいてZバッファリングされるようにする、プロセス。

【請求項115】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを供給するシステムであって、
少なくとも1つのプロセッサと、
少なくとも1つのメモリと、
少なくとも1つの三角描画コマンドを供給するために前記プロセッサと前記メモリに結合された回路と、
を含み、前記三角描画コマンドは、
001111と001011のうちの少なくとも1つに6ビットバイナリ値を有するコマンド識別子フィールドと、

10

20

30

40

50

1セットのエッジ係数と、
 1セットのテクスチャ係数と、
 1セットのバッファ係数と、
 を含み、前記三角描画コマンドフォーマットは、少なくとも1つの三角が前記1セットのエッジ係数にしたがって描画され、少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされ、少なくとも部分的に前記1セットのバッファ係数に基づいてZバッファリングされるようにする、システム。

【請求項116】 3次元グラフィックスシステムにおいて、少なくとも1つの三角描画コマンドを実行するためのプロセスであって、当該三角描画コマンドは、

(a) 001111と001011のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドを解釈することと、

(b) 1セットのエッジ係数を解釈することと、

(c) 1セットのテクスチャ係数を解釈することと、

(d) 1セットのバッファ係数を解釈することと、

(e) 少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされ、少なくとも部分的に前記1セットのZバッファ係数に基づいてZバッファリングされるように、少なくとも1つの三角を前記1セットのエッジ係数にしたがってレンダリングすることと、を含む、プロセス。

【請求項117】 001111と001011のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドを有する、少なくとも1つの三角描画コマンドを実行する3次元グラフィックスシステムであって、

(a) 001111と001011のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドを解釈する手段と、

(b) 1セットのエッジ係数を解釈する手段と、

(c) 1セットのテクスチャ係数を解釈する手段と、

(d) 1セットのZバッファ係数を解釈する手段と、

少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされ、少なくとも部分的に前記1セットのZバッファ係数に基づいてZバッファリングされるように、少なくとも1つの三角を前記1セットのエッジ係数にしたがってレンダリングするための上述の手段に結合される手段と、を含む、システム。

【請求項118】 3次元グラフィックスシステムで用いられる記憶媒体であって、少なくとも1つの三角描画コマンドを記憶し、当該三角描画コマンドは、001111と001011のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットのエッジ係数と、

1セットのテクスチャ係数と、

1セットのバッファ係数と、

を含み、前記三角描画コマンドフォーマットは、少なくとも1つの三角が前記1セットのエッジ係数にしたがって描画され、少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされ、少なくとも部分的に前記1セットのZバッファ係数に基づいてZバッファリングされるようにする、記憶媒体。

【請求項119】 3次元グラフィックスシステムで処理するための、少なくとも1つの表示コマンドを生成するプロセスであって、少なくとも1つの三角描画コマンドを生成するステップを含み、当該三角描画コマンドは、

001010と001110のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットのエッジ係数と、

1セットのテクスチャ係数と、

を含み、前記三角描画コマンドフォーマットは、少なくとも1つの三角が前記1セットのエッジ係数にしたがって描画され、少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされるようにする、プロセス。

【請求項120】 3次元グラフィックスシステムで処理するための、少なくとも1つの表示コマンドを生成するシステムであって、

少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

少なくとも1つの三角描画コマンドを生成するために、前記プロセッサと前記メモリに結合された回路と、を含み、前記三角描画コマンドは、

001010と001110のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットのエッジ係数と、

1セットのテクスチャ係数と、

を含み、前記三角描画コマンドフォーマットは、少なくとも1つの三角が前記1セットのエッジ係数にしたがって描画され、少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされるようにする、システム。

【請求項121】 3次元グラフィックスシステムにおいて、少なくとも1つの三角描画コマンドを実行するプロセスであって、当該三角描画コマンドは、(a) 001010と001110のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドを解釈することと、

(b) 1セットのエッジ係数を解釈することと、

(c) 1セットのテクスチャ係数を解釈することと、

(d) 少なくとも1つの三角を、前記1セットのエッジ係数にしたがって描画し、少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満た

されるようにすることと、
を含む、プロセス。

【請求項122】 6ビットバイナリ値を有するコマンド識別子フィールドを有する、少なくとも1つの三角描画コマンドを実行する3次元グラフィックスシステムであって、

001010と001110のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドを解釈する、コマンド識別子フィールド複号器と、
1セットのエッジ係数を解釈するラスタイザと、
10 1セットのテクスチャ係数を解釈するテクスチャ座標ユニットと、

少なくとも1つの三角を、前記1セットのエッジ係数にしたがって描画し、少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされるようにする表示プロセッサと、を含む、システム。

【請求項123】 少なくとも1つの三角描画コマンドを記憶し、3次元グラフィックスシステムで用いられる記憶媒体であって、当該三角描画コマンドは、001010と001110のうちの少なくとも1つの6ビットバイナリ値を有するコマンド識別子フィールドと、
1セットのエッジ係数と、

1セットのテクスチャ係数と、を含む、前記三角描画コマンドフォーマットは、少なくとも1つの三角が前記1セットのエッジ係数にしたがって描画され、少なくとも部分的に前記1セットのテクスチャ係数に基づいたテクスチャで満たされるようにする、記憶媒体。

【請求項124】 少なくとも1つの三角描画コマンドを生成するためのステップを含み、3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを生成するためのプロセスであって、当該三角描画コマンドは、

001001のセットから選ばれた値の6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットのエッジ係数と、

1セットのZバッファ係数と、を含む、前記三角描画コマンドフォーマットは、少なくとも1つの影のない三角がエッジ係数にしたがって描画され、前記1セットのZバッファ係数の少なくとも一部に基づいてZバッファされるようにする、プロセス。

【請求項125】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを備えるシステムであって、

少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

前記プロセッサおよび少なくとも1つの三角描画コマンドを与えるための前記メモリに接続された回路と、を含み、前記三角描画コマンドは、

001001のセットから選ばれた値の6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットのエッジ係数と、

1セットのZバッファ係数と、を含み、前記三角描画コマンドフォーマットは、少なくとも1つの影のない三角が前記1セットのエッジ係数にしたがって描画され、前記1セットのZバッファ係数の少なくとも一部に基づいてZバッファされるようにする、システム。

【請求項126】 3次元グラフィックスシステムにおいて、少なくとも1つの三角描画コマンドを実行するためのプロセスであって、当該三角描画コマンドは、

(a) 001001のセットから選ばれた値の6ビットバイナリ値を有するコマンド識別子フィールドの解釈と、

(b) 1セットのエッジ係数の解釈と、

(c) 1セットのZバッファ係数の解釈と、

(d) 少なくとも1つの影のない三角を前記1セットのエッジ係数にしたがって描画し、前記1セットのZバッファ係数の少なくとも一部に基づいてZバッファされるようにすること、を含む、プロセス。

【請求項127】 001001のセットから選ばれた値の6ビットバイナリ値を有するコマンド識別子フィールドを有し、少なくとも1つの三角描画コマンドを実行するための3次元グラフィックスシステムであって、前記001001のセットから選ばれた値の6ビットバイナリ値を有するコマンド識別子フィールドの解釈をする複号器と、

1セットのエッジ係数の解釈をするエッジウォークと、

1セットのZバッファ係数の解釈をするバッファコントロールと、

少なくとも1つの影のない三角を前記1セットのエッジ係数にしたがって描画し、前記1セットのZバッファ係数の少なくとも一部に基づいてZバッファされるようにする表示プロセッサと、を含む、システム。

【請求項128】 少なくとも1つの三角描画コマンドを記憶し、3次元グラフィックスシステムに用いる記憶媒体であって、当該三角描画コマンドは、

001001のセットから選ばれた値の6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットのエッジ係数と、

1セットのZバッファ係数と、を含む、前記三角描画コマンドフォーマットは、少なくとも1つの影のない三角が前記1セットのエッジ係数にしたがって描画され、前記1セットのZバッファ係数の少なくとも一部に基づいてZバッファされるようにする、記憶媒体。

【請求項129】 少なくとも1つのテクスチャ矩形描画コマンドを生成するステップを含み、3次元グラフィックスシステムによって処理するための少なくとも1つの表示コマンドを生成するプロセスであって、当該テクスチャ矩形描画コマンドは、

100100および100101の範囲内の6ビットバイナリ値を有するコマンド識別子フィールドと、

少なくとも2つのx座標値と、
 少なくとも2つのy座標値と、
 1セットのテクスチャ係数と、
 タイル記述子インデックス値と、を含み、前記テクスチャ矩形描画コマンドフォーマットは、少なくとも1つの矩形が前記xおよびy座標値にしたがって描画され、前記1セットのテクスチャ係数およびタイル記述子インデックス値の少なくとも一部に基づいたテクスチャで満たされるようにする、プロセス。

【請求項130】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを有するシステムであって、

少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

前記プロセッサおよび少なくとも1つのテクスチャ矩形描画コマンドを備えるための前記メモリに接続された回路と、を含むシステムであって、前記テクスチャ矩形描画コマンドは、

100100および100101の範囲内の6ビットバイナリ値を有するコマンド識別子フィールドと、

少なくとも2つのx座標値と、

少なくとも2つのy座標値と、

1セットのテクスチャ係数と、

タイル記述子インデックス値と、を含み、前記テクスチャ矩形描画コマンドフォーマットは、少なくとも1つの矩形が前記xおよびy座標値にしたがって描画され、前記1セットのテクスチャ係数および前記タイル記述子インデックス値の少なくとも一部に基づいたテクスチャで満たされるようにする、システム。

【請求項131】 3次元グラフィックスシステムにおいて、少なくとも1つのテクスチャ矩形描画コマンドを実行するためのプロセスであって、当該テクスチャ矩形描画コマンドは、

(a) 100100および100101の範囲内の、6ビットバイナリ値を有するコマンド識別子フィールドの解釈と、

(b) 少なくとも2つのx座標値の解釈と、

(c) 少なくとも2つのy座標値の解釈と、

(d) 1セットのテクスチャ係数の解釈と、

(e) タイル記述子インデックス値の解釈と、

(f) 前記xおよびy座標値にしたがって少なくとも1つの矩形のレンダリングし、前記1セットのテクスチャ係数および前記タイル記述子インデックス値の少なくとも一部に基づいたテクスチャで満たされるようにすること、を含む、プロセス。

【請求項132】 100100および100101の範囲内の6ビットバイナリ値を有するコマンド識別子フィールドを有する、少なくとも1つのテクスチャ矩形描画コマンドを実行するための3次元グラフィックスシステムであって、当該システムは、

001001および100101の範囲内の6ビットバイナリ値を有するコマンド識別子フィールドの解釈をする複号器と、

少なくとも2つのx座標値および少なくとも2つのy座標値の解釈をするプロセッサと、

1セットのテクスチャ係数およびタイル記述子インデックス値の解釈をするテクスチャ座標ユニットと、

前記xおよびy座標値にしたがって少なくとも1つの矩形をレンダリングし、前記1セットのテクスチャ係数および前記タイル記述子インデックス値の少なくとも一部に基づいたテクスチャで満たされるようにする表示プロセッサと、を含む、システム。

【請求項133】 少なくとも1つのテクスチャ矩形描画コマンドを記憶し、3次元グラフィックスシステムに用いる記憶媒体であって、当該テクスチャ矩形描画コマンドは、

100100および100101の範囲内の6ビットバイナリ値を有するコマンド識別子フィールドと、

少なくとも2つのx座標値と、

20 少なくとも2つのy座標値と、

1セットのテクスチャ係数と、

タイル記述子インデックス値と、を含み、前記テクスチャ矩形描画コマンドフォーマットは、少なくとも1つの矩形が前記xおよびy座標値にしたがって描画され、前記1セットのテクスチャ係数および前記タイル記述子インデックス値の少なくとも一部に基づいたテクスチャで満たされるようにする、記憶媒体。

【請求項134】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを生成するプロセスであって、

(a) 111010の6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットの色座標と、を含む、少なくとも1セットのプリミティブ色コマンドを生成するステップと、

(b) 110110の6ビットバイナリ値を有するコマンド識別子フィールドと、

少なくとも2つのx座標値と、

少なくとも2つのy座標値と、を含み、満たされたテクスチャ矩形描画コマンドフォーマットは、少なくとも1つの矩形が前記xおよびy座標値にしたがって描画され、前記1セットのテクスチャ係数および色座標の少なくとも一部に基づいた色で満たされるようにする、プロセス。

【請求項135】 3次元グラフィックスシステムによって処理するための少なくとも1つの表示コマンドを生成するシステムであって、

少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

50 (a) 111010の6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットの色座標と、を含む、少なくとも1セットのプリミティブ色コマンドと、

(b) 110110の6ビットバイナリ値を有するコマンド識別子フィールドと、

少なくとも2つのx座標値と、

少なくとも2つのy座標値と、を含む、少なくとも1つの満たされた矩形描画コマンドと、前記満たされた矩形描画コマンドフォーマットは、少なくとも1つの矩形が前記xおよびy座標値にしたがって描画され、前記色座標の少なくとも一部に基づいた色で満たされるようにする、システム。

【請求項136】 3次元グラフィックスシステムにおいて、3次元グラフィックスシステムによって処理するための少なくとも1つの表示コマンドを実行するプロセスであって、

(a) 111010の6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットの色座標と、を含む、少なくとも1セットのプリミティブ色コマンドを解釈するステップと、

(b) 110110の6ビットバイナリ値を有するコマンド識別子フィールドと、

少なくとも2つのx座標値と、

少なくとも2つのy座標値と、を含む、少なくとも1つの満たされた矩形描画コマンドを解釈するステップと、

(c) 少なくとも1つの矩形を前記xおよびy座標値にしたがって描画し、前記色座標の少なくとも一部に基づいた色で満たされるようにするステップと、

を含む、プロセス。

【請求項137】 3次元グラフィックスシステムにおいて、111010の6ビットバイナリ値を有するコマンド識別子フィールドを有する、少なくとも1つの表示コマンドを実行する装置であって、

111010の6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットの色座標と、を含む、少なくとも1セットのプリミティブ色コマンドを解釈する手段と、

110110の6ビットバイナリ値を有するコマンド識別子フィールドと、

少なくとも2つのx座標値と、

少なくとも2つのy座標値と、を含む、少なくとも1つの満たされた矩形描画コマンドを解釈する手段と、

少なくとも1つの矩形を前記xおよびy座標値にしたがって描画し、前記色座標の少なくとも一部に基づいたテクスチャで満たされるようにする手段と、を含む、装置。

【請求項138】 3次元グラフィックスシステムに用いる記憶媒体であって、

(a) 111010の6ビットバイナリ値を有するコマンド識別子フィールドと、

1セットの色座標と、を含む、少なくとも1セットのプ

リミティブ色コマンドと、

(b) 110110の6ビットバイナリ値を有するコマンド識別子フィールドと、

少なくとも2つのx座標値と、

少なくとも2つのy座標値と、を含む、少なくとも1つの満たされた矩形描画コマンドと、を記憶し、前記満たされた矩形描画コマンドフォーマットは、少なくとも1つの矩形が前記xおよびy座標値にしたがって描画され、前記色座標の少なくとも一部に基づいた色で満たされるようにする、記憶媒体。

【請求項139】 少なくとも1つのテクスチャ決定コマンドを生成するステップを有し、3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを生成するプロセスであって、当該テクスチャ決定コマンドは、

110101の6ビットバイナリ値を有するコマンド識別子フィールドと、

画像データフォーマットパラメータと、

色要素サイズパラメータと、

20 タイルラインサイズパラメータと、

開始テクスチャメモリアドレスと、

タイル記述子インデックスと、

バレットナンバーと、

少なくとも1つのテクスチャ座標クランプ可能パラメータと、

少なくとも1つのテクスチャ座標ミラー可能パラメータと、

少なくとも1つのテクスチャ座標ラッピング/ミラーリングマスクと、

30 詳細シフトパラメータの少なくとも1つのテクスチャ座標レベルと、を含む、プロセス。

【請求項140】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを備えるためのシステムであって、

少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

少なくとも1つのテクスチャ決定コマンドを備えるための前記プロセッサと前記メモリに接続された回路と、を含む、当該テクスチャ決定コマンドは、

40 110101の6ビットバイナリ値を有するコマンド識別子フィールドと、

画像データフォーマットパラメータと色要素サイズパラメータと、

タイルラインサイズパラメータと、

開始テクスチャメモリアドレスと、

タイル記述子インデックスと、

バレットナンバーと、

少なくとも1つのテクスチャ座標クランプ可能パラメータと、

50 少なくとも1つのテクスチャ座標ミラー可能パラメータと、

タと、
少なくとも1つのテクスチャ座標ラッピング／ミラーリングマスクと、
詳細シフトパラメータの少なくとも1つのテクスチャ座標レベルと、を含む、システム。

【請求項141】 3次元グラフィックスシステムにおいて、少なくとも1つのテクスチャ決定コマンドを処理するためのプロセスであって、

- (a) 110101の6ビットバイナリ値を有するコマンド識別子フィールドの解釈と、
 - (b) 画像データフォーマットパラメータの解釈と、
 - (c) 色要素サイズパラメータの解釈と、
 - (d) タイルラインサイズパラメータの解釈と、
 - (e) 開始テクスチャメモリアドレスの解釈と、
 - (f) タイル記述子インデックスの解釈と、
 - (g) バレットナンバーの解釈と、
 - (h) 少なくとも1つのテクスチャ座標クランプ可能パラメータの解釈と、
 - (i) 少なくとも1つのテクスチャ座標ミラー可能パラメータの解釈と、
 - (j) 少なくとも1つのテクスチャ座標ラッピング／ミラーリングマスクの解釈と、
 - (k) 詳細シフトパラメータの少なくとも1つのテクスチャ座標レベルの解釈と、
 - (l) 前記ステップの少なくとも一部に基づいた少なくとも1つの画像の生成と、
- を含む、プロセス。

【請求項142】 110101の値の6ビットバイナリ値を有するコマンド識別子フィールドを有する、少なくとも1つのテクスチャ決定コマンドを生成する3次元グラフィックス装置であって、当該装置は、110101の6ビットバイナリ値を有するコマンド識別子フィールドを解釈する手段と、
画像データフォーマットパラメータと、色要素サイズパラメータと、タイルラインサイズパラメータと、開始テクスチャメモリアドレスと、タイル記述子インデックスと、バレットナンバーと、少なくとも1つのテクスチャ座標クランプ可能パラメータと、少なくとも1つのテクスチャ座標ミラー可能パラメータと、少なくとも1つのテクスチャ座標ラッピング／ミラーリングマスクと、詳細シフトパラメータの少なくとも1つのテクスチャ座標レベルと、を解釈する回路と、
前記構造の少なくとも一部に基づいた少なくとも1つの画像を生成する表示プロセッサと、を含む、装置。

【請求項143】 3次元グラフィックスシステムに用いる記憶媒体であって、少なくとも1つのテクスチャ決定コマンドを記憶し、当該テクスチャ決定コマンドは、
110101の値の6ビットバイナリ値を有するコマンド識別子フィールドと、

画像データフォーマットパラメータと、
色要素サイズパラメータと、
タイルラインサイズパラメータと、
開始テクスチャメモリアドレスと、
タイル記述子インデックスと、
バレットナンバーと、
少なくとも1つのテクスチャ座標クランプ可能パラメータと、
少なくとも1つのテクスチャ座標ミラー可能パラメータと、

- 10 少なくとも1つのテクスチャ座標ラッピング／ミラーリングマスクと、
詳細シフトパラメータの少なくとも1つのテクスチャ座標レベルと、を含む、記憶媒体。

【請求項144】 少なくとも1つのテクスチャタイルコマンドを生成するステップを有し、3次元グラフィックスシステムによって処理するための少なくとも1つの表示コマンドを生成するプロセスであって、当該テクスチャタイルコマンドは、

- 20 1セットの110100および110010の範囲内の値の6ビットバイナリ値を有するコマンド識別子フィールドと、
ローおよびハイトイルS座標と、
ローおよびハイトイルT座標と、
タイル記述子インデックスと、を含む、プロセス。

【請求項145】 3次元グラフィックスシステムによって処理するための少なくとも1つの表示コマンドを与えるためのシステムであって、
少なくとも1つのプロセッサと、

- 30 少なくとも1つのメモリと、
少なくとも1つのテクスチャタイルコマンドを与えるための前記プロセッサと前記メモリに接続された回路と、を含む、前記テクスチャタイルコマンドは、
1セットの110100および110010のセットの範囲内の値の6ビットバイナリ値を有するコマンド識別子フィールドと、

ローおよびハイトイルS座標と、
ローおよびハイトイルT座標と、
タイル記述子インデックスと、を含む、システム。

- 40 【請求項146】 3次元グラフィックスシステムにおいて、少なくとも1つのテクスチャタイルコマンドを実行するためのプロセスであって、当該テクスチャタイルコマンドは、

(a) 110100および110010のセットの範囲内の値の6ビットバイナリ値を有するコマンド識別子フィールドの解釈と、

- (b) ローおよびハイトイルS座標の解釈と、
- (c) ローおよびハイトイルT座標の解釈と、
- (d) タイル記述子インデックスの解釈と、
- 50 (e) ステップ(a)～(d)の少なくとも一部に基づい

た表示の生成と、
を含む、プロセス。

【請求項147】 110100および110010の
セットの範囲の値の6ビットバイナリ値を有するコマ
ンド識別子フィールドを有する、少なくとも1つのテク
スチャタイルコマンドを実行する3次元グラフィックス
システムであって、

1セットの110100および110010のセットの
範囲内の値の6ビットのバイナリ値を有するコマンド識
別子フィールドを解釈する復号器と、
ローおよびハイトイルS座標、ローおよびハイトイルT
座標、タイル記述子インデックスの解釈をするテクス
チャユニットと、
座標識別子およびコマンド識別子の少なくとも一部に基
づいた表示を生成する表示プロセッサ回路と、を含む、
システム。

【請求項148】 3次元グラフィックスシステムに用
いる記憶媒体であって、少なくとも1つのテクスチャ
タイルコマンドを記憶し、当該テクスチャタイルコマ
ンドは、

110100および110010のセットの範囲内の値
の6ビットバイナリ値を有するコマンド識別子フィ
ールドと、

ローおよびハイトイルS座標と、

ローおよびハイトイルT座標と、

タイル記述子インデックスと、を含む、記憶媒体。

【請求項149】 少なくとも1つのテクスチャメモ
リブロックコマンドを生成するステップを含み、3次元
グラフィックスシステムによって処理するための、少な
くとも1つの表示コマンドを生成するためのプロセスで
あって、当該テクスチャメモリブロックコマンドは、
110011の値の6ビットバイナリ値を有するコマ
ンド識別子フィールドと、

ローおよびハイトイルS座標パラメータと、

ロータイルT座標パラメータと、

T増大値と、

タイル記述子インデックスと、

を含む、プロセス。

【請求項150】 3次元グラフィックスシステムによ
って処理するための少なくとも1つの表示コマンドを生
成するためのシステムであって、
少なくとも1つのプロセッサと、
少なくとも1つのメモリと、
少なくとも1つのテクスチャメモリブロックローデ
ィングコマンドを生成するための前記プロセッサと前記メ
モリに接続された手段と、を含み、前記テクスチャメモ
リブロックローディングコマンドは、

110011の6ビットバイナリ値を有するコマンド識
別子フィールドと、
ローおよびハイトイルS座標パラメータと、

ロータイルT座標パラメータと、

T増大値と、

タイル記述子インデックスと、を含む、システム。

【請求項151】 3次元グラフィックスシステムにお
いて、少なくとも1つのテクスチャメモリブロックロー
ディングコマンドを実行するためのプロセスであっ
て、当該テクスチャメモリブロックローディングコマ
ンドは、

(a) 110011の6ビットバイナリ値を有するコマ
ンド識別子フィールドの解釈と、

(b) ローおよびハイトイルS座標パラメータの解釈
と、

(c) ロータイルT座標パラメータの解釈と、

(d) T増大値の解釈と、

(e) タイル記述子インデックスと、

(f) ステップ(a)～(e)の少なくとも一部に基づい
た少なくとも1つの構造化プリミティブの表示と、
を含む、プロセス。

【請求項152】 3次元グラフィックスシステムによ
って、110011の6ビットバイナリ値を有するコマ
ンド識別子フィールドを有する、少なくとも1つのテク
スチャメモリブロックローディングコマンドを実行す
るための3次元グラフィックスシステムであって、
110011の6ビットバイナリ値を有するコマンド識
別子フィールドを解釈する手段と、

ローおよびハイトイルS座標パラメータを解釈する手段
と、
ロータイルT座標パラメータを解釈する手段と、
T増大値を解釈する手段と、

タイル記述子インデックスを解釈する手段と、

SおよびT座標パラメータと、T増大値と、タイル記述
子インデックスとの少なくとも一部に基づいた、少なく
とも1つのテクスチャプリミティブを表示する表示回
路と、を含む、システム。

【請求項153】 少なくとも1つのテクスチャメモ
リブロックローディングコマンドを記憶し、3次元グラ
フィックスシステムに用いる記憶媒体であって、当該テ
クスチャメモリブロックローディングコマンドは、
110011の6ビットバイナリ値を有するコマンド識
別子フィールドと、

ローおよびハイトイルS座標パラメータと、

ロータイルT座標パラメータと、

T増大値と、

タイル記述子インデックスと、を含む、記憶媒体。

【請求項154】 3次元グラフィックスシステムによ
って処理するための、少なくとも1つの表示コマンドを
生成するためのプロセスであって、
少なくとも1つのロードテクスチャアルックアップテー
ブルコマンドを生成するステップを含み、当該ロードテ
クスチャアルックアップテーブルコマンドは、

ローおよびハイトイルS座標パラメータと、
ロータイルT座標パラメータと、
T増大値と、
タイル記述子インデックスと、を含む、記憶媒体。

110000の6ビットバイナリ値を有するコマンド識別子フィールドと、

テーブルへのローおよびハイインデックスと、
タイル記述子インデックスと、を含む、プロセス。

【請求項155】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを与えるためのシステムであって、

少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

少なくとも1つのロードテクスチャルックアップテーブルコマンドを備えるための前記プロセッサと前記メモリに接続された回路と、

を含み、前記ロードテクスチャルックアップテーブルコマンドは、

110000の6ビットバイナリ値を有するコマンド識別子フィールドと、

テーブルへのローおよびハイインデックスと、

タイル記述子インデックスと、を含む、システム。

【請求項156】 3次元グラフィックスシステムにおいて、少なくとも1つのロードテクスチャルックアップテーブルコマンドを実行するためのプロセスであって、前記ロードテクスチャルックアップテーブルコマンドは、

(a) 110000の6ビットバイナリ値を有するコマンド識別子フィールドの解釈と、

(b) テーブルへのローおよびハイインデックスの解釈と、

(c) タイル記述子インデックスと、

(d) ステップ(a)～(c)の少なくとも一部に基づいた、メモリへのテクスチャルックアップテーブルのローディングと、

を含む、プロセス。

【請求項157】 110000の値の6ビットバイナリ値を有するコマンド識別子フィールドを有する、少なくとも1つのロードテクスチャルックアップテーブルコマンドを実行するための3次元グラフィックスシステムであって、

110000の値の6ビットバイナリ値を有するコマンド識別子フィールドを解釈するテクスチャメモリ手段と、

テーブルへのローおよびハイインデックスを解釈するための手段と、

タイル記述子インデックスの解釈をするための手段と、
前記コマンド識別子フィールドおよびインデックスの少なくとも一部に基づいたテクスチャメモリへ、テクスチャルックアップテーブルをローディングする手段と、を含む、システム。

【請求項158】 3次元グラフィックスシステムに用いる記憶媒体であって、少なくとも1つのロードテクスチャルックアップテーブルコマンドを記憶し、当該ロ

ードテクスチャルックアップテーブルコマンドは、

110000の6ビットバイナリ値を有するコマンド識別子フィールドと、

テーブルへのローおよびハイインデックスと、

タイル記述子インデックスと、を含む、記憶媒体。

【請求項159】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンド生成するプロセスであって、少なくとも1セットの色コマンドを生成するステップを含み、当該色コマンドは、

110111から111011の範囲内の6ビットバイナリ値を有するコマンド識別子フィールドと、

赤素子パラメータと、

緑素子パラメータと、

青素子パラメータと、

アルファ素子パラメータと、を含む、プロセス。

【請求項160】 詳細小数部フィールドおよび関連最小クランプパラメータの追加レベルを生成するステップをさらに含む、請求項159に記載のプロセス。

【請求項161】 3次元グラフィックスシステムによって処理するための、少なくとも1つの表示コマンドを生成するためのシステムであって、当該システムは、少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

少なくとも1セットの色コマンドを生成するための前記プロセッサおよび前記メモリに接続された回路と、を含み、前記色コマンドは、

110111から111011の範囲の6ビットバイナリ値を有するコマンド識別子フィールドと、

赤素子パラメータと、

緑素子パラメータと、

青素子パラメータと、

アルファ素子パラメータと、を含む、システム。

【請求項162】 詳細小数部フィールドおよび関連最小クランプパラメータの追加レベルを与える回路をさらに含む、請求項161に記載のシステム。

【請求項163】 3次元グラフィックスシステムにおいて、少なくとも1セットの色コマンドを解釈するためのプロセスであって、当該色コマンドは、

(a) 110111から111011の範囲の6ビットバイナリ値を有するコマンド識別子フィールドの解釈と、

(b) 赤素子パラメータの解釈と、

(c) 緑素子パラメータの解釈と、

(d) 青素子パラメータの解釈と、

(e) アルファ素子パラメータの解釈と、

(f) ステップ(a)～(e)の少なくとも一部に基づいた画像の生成と、

を含む、プロセス。

【請求項164】 詳細小数部フィールドおよび関連最小クランプパラメータの追加レベルを解釈するステップ

をさらに含む、請求項163に記載のプロセス。

【請求項165】 110111から111011の範囲の6ビットバイナリ値を有するコマンド識別子フィールドを有する、少なくとも1セットの色コマンドを解釈するための3次元グラフィックスシステムであって、110111から111011の範囲の6ビットバイナリ値を有するコマンド識別子フィールドを解釈する手段と、

赤素子パラメータを解釈する手段と、

緑素子パラメータを解釈する手段と、

青素子パラメータを解釈する手段と、

アルファ素子パラメータを解釈する手段と、

前記パラメータの少なくとも一部に基づいたイメージを生成する表示回路と、を含む、システム。

【請求項166】 詳細小数部フィールドおよび関連最小クランプパラメータの追加レベルを解釈する手段をさらに含む、請求項165に記載のシステム。

【請求項167】 3次元グラフィックスシステムに用いる記憶媒体であって、当該記憶媒体は少なくとも1セットの色コマンドを記憶し、当該色コマンドは、

110111から111011の範囲の6ビットバイナリ値を有するコマンド識別子フィールドと、

赤素子パラメータと、

緑素子パラメータと、

青素子パラメータと、

アルファ素子パラメータと、を含む、記憶媒体。

【請求項168】 詳細小数部フィールドおよび関連最小クランプパラメータの追加レベルを記憶する手段をさらに含む、請求項167に記載の記憶媒体。

【請求項169】 少なくとも1セットのプリミティブ奥行きコマンドを生成するステップを含み、3次元グラフィックスシステムによって処理するための少なくとも1つの表示コマンドを生成するためのプロセスであって、当該プリミティブ奥行きコマンドは、

101110の6ビットバイナリ値を有するコマンド識別子フィールドと、

プリミティブ奥行きパラメータと、

プリミティブデルタ奥行きパラメータと、を含む、プロセス。

【請求項170】 3次元グラフィックスシステムによって処理するための少なくとも1つの表示コマンドを与えるためのシステムであって、当該システムは、

少なくとも1つのプロセッサと、

少なくとも1つのメモリと、

少なくとも1セットのプリミティブ奥行きコマンドを与えるための前記プロセッサと前記メモリに接続された手段と、を含み、前記プリミティブ奥行きコマンドは、

101110の6ビットバイナリ値を有するコマンド識別子フィールドと、

プリミティブ奥行きパラメータと、

プリミティブデルタ奥行きパラメータと、を含む、システム。

【請求項171】 3次元グラフィックスシステムにおいて、少なくとも1セットのプリミティブ奥行きコマンドを処理するためのプロセスであって、当該プリミティブ奥行きコマンドは、

(a) 101110の6ビットバイナリ値を有するコマンド識別子フィールドの解釈と、

(b) プリミティブ奥行きパラメータの解釈と、

10 (c) プリミティブデルタ奥行きパラメータの解釈と

(d) ステップ(a)～(c)の少なくとも一部に基づいた少なくとも1つの画像の生成と、

を含む、プロセス。

【請求項172】 101110の6ビットバイナリ値を有するコマンド識別子フィールドを有する、少なくとも1セットのプリミティブ奥行きコマンドを処理するための3次元グラフィックスシステムであって、当該システムは、

101110の6ビットバイナリ値を有するコマンド識別子フィールドを解釈する手段と、

20 プリミティブ奥行きパラメータを解釈する手段と、

プリミティブデルタ奥行きパラメータを解釈する手段と、

と、

パラメータの少なくとも一部に基づいた画像を生成する表示回路と、

を含む、システム。

【請求項173】 少なくとも1セットのプリミティブ奥行きコマンドを記憶し、3次元グラフィックスシステムに用いる記憶媒体であって、当該プリミティブ奥行きコマンドは、

101110の6ビットバイナリ値を有するコマンド識別子フィールドと、

プリミティブ奥行きパラメータと、

プリミティブデルタ奥行きパラメータと、を含む、記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、低コストのビデオゲームシステムに関する。より詳細には、本発明は、3次元でワールドをモデル化でき、かつ、変更可能な視点に基づいて選択された2次元ビュー平面上に、当該モデルを投影することができるビデオゲームシステムに関する。

【0002】[本発明の背景および要約]人間の想像力は、視覚画像により刺激される。実際に、日没時に見られるもの、夜に夢みるもの、小説を読んだ時に心に描く心像等、これらの忘れられないシーンの全てが、視覚画像から構成されている。歴史の至る所で、人々は、これらの画像を鉛筆あるいは絵具で、または、ビデオテープで記録しようとした。しかし、コンピュータの出現によ

り、人々は、画像が現実の世界で、あるいは想像力の中で表現されたものと同じ鮮明さ、詳細さ、および現実感をもってそれらの画像を生成することができるようになり始めた。

【0003】任天堂エンターテインメントシステムおよびスーパー任天堂エンターテインメントシステムのようなコンピュータベースのホームビデオゲーム機は、エキサイティングなビデオグラフィックスを対話的に生成できることから、非常に成功を収めている。しかし、これらの従来のビデオグラフィックシステムは、追加の増設ハードウェアを必要とせずに、一般に2次元で動作され、平らな紙の切り抜きを掲示板に留めるのと多少似た方法で、フラット（平面）画像表現によりグラフィック表示を生成する。非常にエキサイティングなゲームプレイが、2次元グラフィック法を用いて生成できるが、2Dシステムは、3次元グラフィックシステムにより実現されるリアリズムを提供することができない。

【0004】3Dグラフィックシステムは、基本的に2Dグラフィックスとは異なる。3Dグラフィック法では、「ワールド」は3次元空間で表される。このシステムは、ユーザがワールド内で視点を選択することを可能にする。このシステムは、選択された視点に基づいてワールドを「投影する」ことにより、画像を生成する。この結果は、奥行きおよびリアリズムを備えた真の3次元画像である。

【0005】幾年にも渡って、専門家は、信じられないほどリアルな3D画像、例えば、車、飛行機、および分子の超精細モデル（ジェット戦闘機の操縦席あるいは五輪のボブスレーのフロントシートから見たようなヴァーチャリアリティ）および「ジュラシックパーク」の恐竜を生成するために、スーパーコンピュータおよびハイエンドワークステーションを使用していた。しかし、これまでは、コンピュータシステムは、そのような画像を対話的に生成するために数万ドルもの費用（これは平均的な顧客の手の届く範囲を優に超える）を必要とした。

【0006】ここに開示されている低コスト高性能3Dグラフィックシステムは、十分に特色のある低コスト高性能システムによって、専門家だけでなく多数のゲームプレイヤーに、素晴らしい3Dワールドの内部に対話する機会を初めて与えることを意図している。プレーヤが得るものは、いかなる家庭用コンピュータシステムのパワーの数倍も本当に素晴らしい、より現実的な3次元アニメーション、素敵なグラフィックスである。これらの全てが、平均的な顧客の手の届く範囲内にあるように十分に低いコストで供給される。

【0007】以下に、本発明によるシステムにより実現される多くの有利な特徴の幾つかの例を示す。

- ・低価格システムの現実的対話式3Dグラフィックス
- ・低コストシステムのビデオゲームプレイおよび他のグラフィックスアプリケーションを提供し、および/また

は特定のスクリーン効果を生成するために、カラーテレビ受像機に使用する低価格システム用の最適特徴セット／アーキテクチャ

- ・高性能3Dグラフィックスおよびデジタル音響処理を提供するコプロセッサ

- ・低コストカラーテレビジョン規準システムにおいて、高品質ステレオ音響および3Dグラフィックスを実現する、グラフィックスデジタル処理とオーディオ信号処理とに共用される信号プロセッサ

10 ・統合RAM法が柔軟性を増加すること

- ・全ての主要システム要素が共用RAMを介して通信すること

- ・狭いメインメモリバス幅を補償する手法／構成

- ・記憶装置（例えば、携帯メモ리카ートリッジ）から実行できるコードが、コモンRAMにロードでき、コプロセッサメモリアクセス／結合回路を介してメインプロセッサによりアクセスできること

20 ・マイクロコード記憶装置にロードできるグラフィックコプロセッサが、更なる柔軟性を与え、かつ互換性の問題点を簡単化するために、携帯記憶媒体からマイクロコードを受信する

- ・マイクロコードが「ブートROM」命令の実行によりロードされること

- ・コプロセッサ内にグラフィックス機能およびオーディオ機能と呼出し、かつグラフィックスコプロセッサとシステムの他の部分との間にインタフェイスを与えるために、最適なコマンドおよび関連フォーマットが使用されること

30 ・特定のハードウェアレジスタの定義、フォーマットおよび関連の機能を有するコプロセッサレジスタのセット

- ・マイクロコードのグラフィックスおよびオーディオ構成／処理が効果的な高性能動作を提供する

- ・ベクトルユニットが、低コストパッケージでのグラフィックスおよびオーディオデジタル処理に対して最適性能を与えること

- ・バイブライনラスタライズエンジンが、サイクル毎1画素モードおよびサイクル毎2画素モードを提供し、ハードウェアコストを最小にし、また豊富な特徴セットを提供すること

40 ・コプロセッサの少ないピン出力

【0008】〔好適な実施例の詳細な説明〕図1は、本発明に係るビデオゲームシステム50の実施例を示している。この実施例のビデオゲームシステム50は、メインユニット52、ビデオゲーム記憶装置54、およびハンドホールドコントローラ56（あるいは他のユーザ入力装置）から構成されている。この実施例では、メインユニット52は、通常の家計用カラーテレビ受像機58に接続されている。カラーテレビ受像機58は、そのテレビ画面上に3Dビデオゲーム画像を表示し、かつそのスピーカ62によりステレオ音響を再生する。

【0009】この実施例では、ビデオゲーム記憶装置54は、メインユニット52の上面66に設けられたスロット64に挿入できる取外し可能なメモリカートリッジの形をとっている。ビデオゲーム記憶装置54は、例えば、内部に読取り専用メモリ（ROM）チップ76を収納した、プラスチックのハウジング68から構成されている。読取り専用メモリ76は、この実施例では、ビデオゲームソフトウェアを格納している。ビデオゲーム記憶装置54がメインユニットスロット64に挿入されると、カートリッジの電気接点74は、メインユニット内の「エッジコネクタ」の電気接点と噛み合う。これにより、ビデオゲーム記憶装置54内の読取り専用メモリ76が、メインユニット52内の電子回路に電氣的に接続される。

【0010】「読取り専用メモリ」チップ76は、特定のビデオゲームに関するソフトウェア命令および他の情報を記憶している。ビデオゲーム記憶装置54内の読取り専用メモリチップ76は、例えばアドベンチャーゲーム用の命令および他の情報を格納している。別のビデオゲーム記憶装置54内の読取り専用メモリチップ76は、ドライビングゲームつまりカーレースゲームをするための命令および情報を格納している。更に別のビデオゲーム記憶装置54内の読取り専用メモリチップ76は、教育ゲームをするための命令および情報を格納している。全く別のゲームをするためには、ビデオゲームシステム50のユーザは、単に適切なビデオゲーム記憶装置54をメインユニットスロット64に差し込めばよく、これによりビデオゲーム記憶装置54内の読取り専用メモリチップ76（および任意の他の回路の記憶装置も含まれる）が、メインユニット52に接続される。これにより、メインユニット52が読取り専用メモリチップ76内に格納された情報にアクセスすることが可能になる。この情報は、読取り専用メモリ内のビデオゲームソフトウェアの制御の下で、特定された画像をカラーテレビ受像機58に表示し、かつその音響を再生することにより、適当なビデオゲームを行うようにメインユニット52を制御する。

【0011】ビデオゲームシステム50によりビデオゲームをするには、ユーザは、まずケーブル78を接続することにより、メインユニット52をカラーテレビ受像機58に接続する。メインユニット52は、カラーテレビ受像機58を制御する「ビデオ」信号と「オーディオ」信号とを出力する。「ビデオ」信号は、テレビ画面60上に表示される画像を制御するものであり、一方「オーディオ」信号は、テレビスピーカ62により音響として再生される。カラーテレビ受像機58の形式によっては、メインユニット52とカラーテレビ受像機58とのライン間に、「RF変調器」というユニットをさらに使用することが必要である。「RF変調器」（図示せず）は、メインユニット52のビデオおよびオーディオ

出力を、放送形式のテレビ信号に変換する。このテレビ信号は、テレビ受像器内部の「チューナ」により受信され処理される。

【0012】ユーザは、メインユニット52を電源に接続することも要求される。この電源は、標準家庭用の電気ソケットに差込まれて、家庭用電源をメインユニット52を動作させるのに適した低DC電圧に変換する、通常のACアダプタ（図示せず）を内蔵している。

【0013】次に、ユーザは、ハンドコントローラ56a、56bをメインユニットのフロントパネル82上の対応するコネクタ80に接続する。ハンドコントローラ56は、各種の形式をとることができる。この実施例で示されたコントローラ56は、押しボタン84および方向スイッチあるいは他のコントロール86を備えている。方向スイッチ86は、例えば、テレビ画面60上に表示されるキャラクタが移動すべき方向（上、下、左あるいは右）を特定するため、および／または3Dワールドの中の視点を特定するために使用される。他の可能なものには、例えば、ジョイスティック、マウスポインタコントロール、および他の従来のユーザ入力装置が含まれる。この実施例では、コントローラ56は、4人プレイのゲームを可能にするために、4つまでメインユニット52に接続できる。

【0014】次に、ユーザは、自分が楽しみたいビデオゲームを格納しているビデオゲーム記憶装置54を選択し、そのビデオゲーム記憶装置54をメインユニットのスロット64に挿入する（これにより、プリント回路基板70および関連するエッジ接点74を介して、読取り専用メモリ76をメインユニットの電子回路に接続する）。次に、ユーザは、電源スイッチ88を操作し、ビデオゲームシステム50をオンにする。これにより、メインユニット52が、読取り専用メモリ76に格納されたソフトウェアに基づいて、ビデオゲームのプレイを開始する。ユーザは、コントローラ86を操作して、メインユニット52に入力を与えて、ビデオゲームのプレイを実行する。例えば、ユーザが、押しボタン84の一つを押すことにより、ゲームが開始される。前述したように、方向スイッチ86を操作することにより、アニメーションのキャラクタがテレビ画面60上を異なる方向に動き、あるいは3Dワールド内のユーザの視点に変化する。ビデオゲーム記憶装置54内に記憶された特定のビデオゲームによっては、コントローラ56上の種々のコントロール84、86は違う時間に違う機能を実行できる。ユーザが、ゲームのプレイを再スタートしたい場合には、ユーザは、リセットボタン90を押すことができる。

【0015】〔3Dスクリーン効果の実施例〕ビデオゲームシステム50は、ワールド（あるいはその一部）をこのワールド内の任意の視点から表示するように、デジタル表示あるいは3次元ワールドのモデルを対話式にリ

アルタイムで処理できる。例えば、ビデオゲームシステム50は、ゲームコントローラ86からのリアルタイム入力に応答して対話式に視点を変更できる。このため、例えば、ワールド内を動き、かつゲームプレイヤーが行くように命令するあらゆる所を見てそこに行くという「仮想の人物」の視点を通して、ゲームプレイヤーは、ワールドを見ることができる。対話式にリアルタイムで3D画像の特性を表示するこの能力は、極めてリアルでエキサイティングなゲームプレイを創り出す。

【0016】図2(a)～図3(f)は、ビデオゲームシステム50が、カラーテレビ受像機58の画面上に創り出すことができる3次元スクリーン効果の単なる一つの例を示している。特許書類がカラーでプリントできないことから、図2(a)～図3(f)は白黒で示されているが、ビデオゲームシステム50は、これらの異なる画面をカラーテレビ受像機上に明るいカラーで表示できる。更に、ビデオゲームシステム50は、ゲームコントローラ86の操作に応答して、これらの画像をリアルタイムで極めて迅速に(例えば、数秒あるいは数十秒で)創り出すことができる。

【0017】図2(a)～図3(f)の各々は、丘の頂上にあるお城を表現している「ワールド」の3次元モデルを用いて発生されたものである。このモデルは、幾何学的形状(つまり、多角形)と、この幾何学的形状により確定された形状表面上に「マッピング」される「テクスチャ」(デジタルに記憶された画像)とから構成されている。ビデオゲームシステム50は、適切にこれらの幾何学的形状のサイズを決め、回転し、移動し、それらを「投影し」、そしてそれらを一体化して、任意の視点からみた3次元ワールドの現実的な画像を創り出す。ビデオゲームシステム50は、ゲームコントローラ86によるユーザの操作に応答して、対話式にリアルタイムでこのことを実行できる。

【0018】図2(a)～図2(c)および図3(f)は、4つの異なる視点から見たお城の空からの図を示している。各図は斜視図であることに注目してもらいたい。ビデオゲームシステム50は、認識できる遅延なくあるいは少しの遅延をもって、数秒でインタラクティブにこれらの空からの図(およびそれらの間の図)を発生でき、そのためあなたもビデオゲームプレイヤーが、実際に

お城の上空を飛んでいるかのようにこれらの空からの図が現れる。

【0019】図2(d)および図2(e)は、お城のメインゲートで、あるいはその近くで地面から見上げた図を示している。ビデオゲームシステム50は、視点をお城の前に「着陸」するように命令し、かつ「仮想の観察者」(つまり、画面がその人物の目を通して表示されるという、3Dワールドを移動する仮想の人物)を別の方向に向くように命令するゲームコントローラの入力に応答して、対話式にリアルタイムでこれらの図を発生す

る。図2(d)は、非常にリアルな画像を創り出すために、煉瓦壁のテクスチャ(ピクチャ)がお城の壁にマッピングされるという「テクスチャマッピング」の例を示している。

【0020】[ビデオゲームシステムの全体エレクトロニクス]図4は、メインユニット52内の主要な電子回路は、メインプロセッサ100と、コプロセッサ200と、およびメインメモリ300とを有していることを示している。メインプロセッサ100は、コントローラ56により与えられた入力に基づいて、ビデオゲーム記憶装置54に供給されたビデオゲームプログラムを走らせるコンピュータである。コプロセッサ200は、メインプロセッサ100から得た命令およびコマンドに基づいて、画像および音響を発生する。メインメモリ300は、メインプロセッサ100およびコプロセッサ200が動作するのに必要な情報を記憶する高速メモリであり、かつメインプロセッサ100およびコプロセッサ200とで共用されている。この実施例では、メインメモリ300への全てのアクセスは、コプロセッサ200を介して行われる。

【0021】この実施例では、メインプロセッサ100は、メインプロセッサ100とコプロセッサ200との間の通信バス102により、コプロセッサ200を介し、ビデオゲームプログラムにアクセスする。メインプロセッサ100は、コプロセッサ200とビデオゲーム記憶装置54との間の別の通信バス104を介して、ビデオゲーム記憶装置54から読み出すことができる。メインプロセッサ100は、ビデオゲームプログラムをバス106によりビデオゲーム記憶装置54からメインメモリ300にコピーでき、次にコプロセッサ200およびバス102、106を介して、メインメモリ300内のビデオゲームプログラムにアクセスできる。

【0022】メインプロセッサ100は、コプロセッサ200に何を成すべきかを指示するコマンドのリストを時々発生する。この実施例では、コプロセッサ200は、3Dグラフィックスおよびデジタルオーディオを迅速に処理するために、最適化された内部デザインを有する専用高性能アプリケーションである特定集積回路(ASIC)を備えている。コプロセッサ200は、バス102を介してメインプロセッサ100により与えられたコマンドに応答して、カラーテレビ受像機58へ供給するビデオおよびオーディオ信号を発生する。コプロセッサ200は、メインメモリ300および/またはビデオゲーム記憶装置54に記憶されたグラフィックスデータ、オーディオデータおよび他のデータを使用して、画像および音響を発生する。

【0023】図4は、この実施例では、コプロセッサ200が、信号プロセッサ400および表示プロセッサ500を備えていることを示している。信号プロセッサ400は、ビデオゲーム記憶装置54から供給される「マ

マイクロコード」コンピュータプログラムの制御下で、グラフィックスの幾何学的処理およびデジタルオーディオ信号の処理を実行する、埋め込みプログラム可能マイクロコントローラである。表示プロセッサ500は、グラフィックスプリミティブを与え、これにより、カラーテレビ受像機58上に表示する画像を創り出す高速状態マシンである。信号プロセッサ400および表示プロセッサ500は、別個に動作するが、信号プロセッサ400は、グラフィックスコマンドを送ることにより表示プロセッサ500を監視できる。信号プロセッサ400および表示プロセッサ500は、共にメインプロセッサ100により直接制御できる。以下は、信号プロセッサ400および表示プロセッサ500が実行できる機能および動作の例である。

【0024】信号プロセッサ

- ・マトリクス制御
- ・3D変換
- ・ライティング
- ・クリッピング、遠近法およびビューポートの適用
- ・表示プロセッサのコマンド発生

表示プロセッサ

- ・ラスタライゼーション
- ・テクスチャの座標発生
- ・テクスチャの適用およびフィルタリング
- ・色の組み合わせ
- ・ブレンド
- ・フォギング
- ・アンチエイリアジング
- ・フレームバッファおよびフレームバッファ制御

【0025】図5は、この実施例のビデオゲームシステム50におけるメインプロセッサ100、コプロセッサ200、およびメインメモリ300により実行される主要な処理を示している。メインプロセッサ100は、ゲームコントローラ56から入力を受け取り、ビデオゲーム記憶装置54により与えられたビデオゲームプログラムを実行してゲームの処理を行う（ブロック120）。これはアニメーションを与え、コプロセッサ200で使用するグラフィックスおよびオーディオコマンドを組み立てる。メインプロセッサ100により発生されたグラフィックスおよび音響コマンドは、ブロック122、124、および126（これら各々は、コプロセッサ200により実行される）により処理される。この実施例では、信号プロセッサ400は、3D幾何学変換およびライティング処理を実行して（ブロック122）、表示プロセッサ500へのグラフィックス表示コマンドを発生する。表示プロセッサ500は、グラフィックスのプリミティブ（例えば、線、三角、および矩形）を「描き」、カラーテレビ受像機58上に表示する画像を創り出す。表示プロセッサ500は、各プリミティブを「ラスタライズ」することにより、更に、必要ならば、これ

にテクスチャを貼り付けることにより、この「描画」機能あるいはレンダリング機能を実行する（ブロック126）。表示プロセッサ500は、この処理を非常に迅速に、例えば、毎秒数百万「画素（カラーテレビの画像の要素）」の速さで行う。表示プロセッサ500は、その画像出力をメインメモリ300内のフレームバッファに書き込む（ブロック128）。このフレームバッファは、テレビ画面60上に表示されるべき画像のデジタル表示を記憶する。コプロセッサ200内の別の回路は、このフレームバッファから情報を読み取り、これをカラーテレビ受像機58に表示するために出方する（ブロック130）。

【0026】信号プロセッサ400は、またデジタルオーディオ信号処理法を用いて、メインプロセッサ100から入力された音響コマンドも処理する（ブロック124）。信号プロセッサ400は、デジタルオーディオ出力をメインメモリ300内の音響バッファに書き込む。メインメモリ300は、この音響出力を一時的に「バッファ（つまり記憶）」する（ブロック132）。コプロセッサ200内の他の回路は、メインメモリ300からこのバッファされた音響データを読み取り、これを電気オーディオ信号（ステレオ左および右チャンネル）に変換し、テレビスピーカ62a、62bに送り再生する（ブロック134）。

【0027】カラーテレビ受像機58は、毎秒30あるいは60枚の新しい画像を表示する。この「フレーム速度」は、人間の目をごまかして連続動作を見ているように思わせ、これにより、メインユニット52がフレーム毎に画像を少しずつ変化させることで、テレビ画面60上にアニメーション効果を発生できる。このテレビのフレーム速度を維持するために、コプロセッサ200は、毎秒1/30あるいは1/60の新しい画像を創り出さねばならない。コプロセッサ200は、テレビ画面60上のアニメーション効果と共に、一連の連続音響も発生できる。

【0028】[全体システムの動作] 図6は、ビデオゲームシステム50の全体動作をより詳細に示しており、図7は、グラフィックスを発生するためにビデオゲームシステム50により実行される全ステップを示している。この実施例では、メインプロセッサ100は、メインメモリ300に記憶されたビデオゲームプログラム108を読みとる（一般に、このビデオゲームプログラム108は、もともとビデオゲーム記憶装置54に格納されており、このビデオゲーム記憶装置54からメインメモリ300にコピーされたものである）。ビデオゲームプログラム108を実行することに応答して（およびゲームコントローラ56からの入力に応答して）、メインプロセッサ100は、コプロセッサ200へのコマンドのリスト110を創り出す（あるいは、カラーテレビ受像機58から読み出す）（図7、ブロック120a）。

このリスト110は、一般に二種類のコマンドを含んでいる。

(1) グラフィックスコマンド

(2) オーディオコマンド

グラフィックスコマンドは、どのような画像をテレビ画面60上に発生すべきかをコプロセッサ200に指示する。オーディオコマンドは、テレビスピーカ62で再生するために、どのような音響を発生すべきかをコプロセッサ200に指示する。

【0029】グラフィックスコマンドのリストは、コプロセッサ200がテレビ画面60上に表示するものを制御することから「表示リスト」と呼ばれる。オーディオコマンドのリストは、スピーカ62により再生される音響を制御することから「再生リスト」と呼ばれる。一般に、メインプロセッサ100は、カラーテレビ受像機58の各ビデオ「フレーム」時間に対して、新しい表示リストおよび新しい再生リストを特定する。

【0030】この実施例では、メインプロセッサ100は、上記リストをメインメモリ300に記憶し、次にコプロセッサ200にその記憶位置を知らせることにより、コプロセッサ200へ表示／再生リスト110を供給する(図7、ブロック120c)。メインプロセッサ100は、また、コプロセッサ200が表示／再生リスト110で要求されたグラフィックスおよびオーディオを発生するために必要とされる全てのデータを格納しているグラフィックおよびオーディオデータベース112が、メインメモリ300に備えられていることを確認する。グラフィックおよびオーディオデータベース112のあるものあるいは全ては、ビデオゲーム記憶装置54から取り出すことができる。表示／再生リスト110は、コプロセッサ200がグラフィックおよびオーディオデータベース112のどの部分を使用すべきかを特定する。メインプロセッサ100は、信号プロセッサ400が「マイクロコード(つまり、信号プロセッサに何をすべきかを指示するコンピュータプログラム)」をロードしたことを確認することにも対応できる。

【0031】信号プロセッサ400は、メインプロセッサ100から表示／再生リスト110を読み取り(図7、ブロック122a)、このリストを処理する(必要に応じて、データベース112内の別のデータにアクセスする)(図7、ブロック122b)。信号プロセッサ400は、二つの主な出力、つまり表示プロセッサ500による更なる処理のためのグラフィックス表示コマンド112と(図7、ブロック122c)、メインメモリ300内に一時記憶するためのオーディオ出力データ114とを発生する。信号プロセッサ400は、スピーカ62によってオーディオを再生するために要する時間よりも非常に短い時間でオーディオデータを処理する。

「オーディオインタフェイス」(図示せず)と呼ばれるコプロセッサ200の別の部分は、バッファされたオー

ディオデータを引き続いて読み取り、テレビスピーカ62による再生のためにそのオーディオデータをリアルタイムで出力する。

【0032】信号プロセッサ400は、コプロセッサ200の内部バスを介してグラフィック表示コマンド112を直接表示プロセッサ500に供給でき、あるいは表示プロセッサ(図示せず)による検索のために、これらのグラフィック表示コマンドをメインメモリ300に書き込む。これらのグラフィック表示コマンド112は、

特定された特性によって特定された幾何学形状を描く(「レンダリングする」)ことを表示プロセッサ500に命令する(図7、ブロック126a)。例えば、表示プロセッサ500は、これらのグラフィック表示コマンド112に基づいて線、三角形、あるいは矩形(多角形)を描くことができ、また、全てグラフィック表示コマンド112により特定されたように、三角形および矩形を特定の色で塗り、および／またはテクスチャ116(例えば、木の葉あるいは煉瓦塀の煉瓦の画像)を貼り付けることができる。メインプロセッサ100は、表示プロセッサ500によるアクセスのために、テクスチャ画像116をメインメモリ300に記憶する。表示プロセッサ500に直接命令するために、メインプロセッサ100は、表示プロセッサ500による検索のためのグラフィック表示コマンド112を、メインメモリ300に直接書き込むこともできる。

【0033】表示プロセッサ500は、テレビ画面60上に現れるべき画像のデジタル表示をその出力として発生する(図7、ブロック126b)。このデジタル画像は、しばしば「ビットマップ」と呼ばれ、メインメモリ300内に存在するフレームバッファ118内に記憶される。表示プロセッサ500は、画像の奥行き情報を記憶するために、メインメモリ300内のフレームバッファ118bを使用し記憶する。「ビデオインタフェイス」(図示せず)と呼ばれるコプロセッサ200の別の部分は、フレームバッファ118を読み取り、その内容をカラーテレビ受像機58に供給するビデオ信号に変換する(図7、ブロック127)。一般的には、フレームバッファ118は、「ダブルバッファ」される。ダブルバッファとは、コプロセッサ200がフレームバッファ118の半分に「次の」画像を書き込み、一方、ビデオインタフェイスがフレームバッファ118の他の半分を読み出すことを意味する。

【0034】図7に、前述された種々のステップが、この実施例では、「パイプライン」処理されることを示す。「パイプライン」は、異なる動作が同時にグラフィックス発生処理の異なる段階で実行されることを意味する。簡単な例は、大勢の人たちで洗濯をするその仕方である。非パイプラインモードで洗濯するということは、次のかたまりの洗濯物を始める前に、ひとかたまりの洗濯物をするための全ての関連作業(洗う、乾燥する、ア

10

20

30

40

50

イロンをかける、折り畳む、そして片づける)を実行することを意味する。時間を節約するために、たくさんの洗濯物をする人たちは、複数のかたまりの洗濯物に対して、洗う、乾燥する、アイロンをかける、折り畳む、そして片づけるという動作を同時に実行することにより、洗濯の過程を「パイプライン」処理する。

【0035】同様に、メインプロセッサ100、信号プロセッサ400、表示プロセッサ500、およびビデオインタフェイス210により実行される動作が、この実施例では「パイプライン」処理される。例えば、この実施例では、メインプロセッサ100は、前もって表示リストを二つのビデオフレームに組立てることができ、一方、信号プロセッサ400および表示プロセッサ500は、前もってひとつのビデオフレームについてデータを処理でき、また、ビデオインタフェイス210は、進行中の現在のビデオフレームについてのデータを処理する。以下に説明するように、ブロック126aで表示プロセッサ500により実行される詳細なグラフィックスレンダリングステップも、速度性能を最大にするためにパイプライン処理される。

【0036】[より詳細なシステムアーキテクチャ]図8は、ビデオゲームシステム50のより詳細なアーキテクチャを示している。この図は、ビデオゲームメインユニット52を示している。ビデオゲームメインユニット52は、メインプロセッサ100、コプロセッサ200、およびメインメモリ300に加えて、クロック発生器136、シリアル周辺インタフェイス138、オーディオデジタル/アナログ変換器(DAC)140、オーディオ増幅器/ミキサ142、ビデオデジタル/アナログ変換器144、およびビデオエンコーダ146等の追加の要素を備えている。

【0037】この実施例では、クロック発生器136(これは水晶振動子148により制御される)は、メインユニット52の他の要素のタイミングおよび同期をとるタイミング信号を発生する。異なるメインユニット要素は、異なるクロック周波数を要求し、またクロック発生器136は、適切なクロック周波数出力(あるいは分周すること等により、適切なクロック周波数が得られるような周波数)を与える。コプロセッサ200内のブロック216は、クロック発生器136からクロック信号を受け、(必要ならば適当に分周した後)それらのクロック信号をコプロセッサ200内のそれぞれの他の回路に分配する。

【0038】この実施例では、ゲームコントローラ58は、メインプロセッサ100には直接には接続されていないが、かわりにシリアル周辺インタフェイス138を介してメインユニット52に接続されている。シリアル周辺インタフェイス138は、4台(あるいは5台)までのゲームコントローラ56(あるいは他のシリアル周辺装置)から入力されるシリアルデータ信号をデマルチ

プレクスし、このデータを所定のフォーマットで、コプロセッサ200を介してメインプロセッサ100に与える。この実施例では、シリアル周辺インタフェイス138は、双方向性である、つまりシリアル周辺インタフェイス138は、シリアル情報を受信すると共にメインプロセッサ100により特定されたシリアル情報を送信できる。

【0039】この実施例では、シリアル周辺インタフェイス138は、少量のインシヤルプログラムロード(IPL)コードを記憶する「ブートROM」読取り専用メモリ150も有している。「ブートROM」150内に記憶されたこのIPLコードは、メインプロセッサ100がビデオゲーム記憶装置54内のインシヤルゲームプログラム命令108aの実行を開始できるようにするために、スタート時および/またはリセット時にメインプロセッサ100によって実行される(図9、ブロック160a、160b参照)。インシヤルゲームプログラム命令108aは、次にメインプロセッサ100を制御し、メインメモリ300をアクセスするために必要なドライバおよびコントローラを初期化し(図9、ブロック160c、160d参照)、また、メインプロセッサ100およびコプロセッサ200により実行し使用するために、ビデオゲームプログラムおよびデータをより高速のメインメモリ300にコピーする(図9、ブロック160e、160f、160g参照)。

【0040】また、この実施例では、シリアル周辺インタフェイス138は、ビデオゲーム記憶装置54内の関連のセキュリティプロセッサ152(例えば、小さいマイクロプロセッサ)と通信するセキュリティプロセッサ(例えば、別の小さいマイクロプロセッサ)を備えている(図8参照)。このセキュリティプロセッサ152の対(一方はビデオゲーム記憶装置54内にあり、他方はメインユニット52内にある)は、許可された記憶装置だけがビデオゲームメインユニット52と共に使用できることを保証するという認証機能を実行する。米国特許第4,799,635号を参照。この実施例では、シリアル周辺インタフェイス138内のセキュリティプロセッサ152は、ソフトウェアの制御下でセキュリティ機能を実行することに加えて、ゲームコントローラ56から受けたデータをソフトウェアの制御下で処理する。

【0041】図8は、ビデオゲームメインユニット52内のコネクタ154を示している。この実施例では、このコネクタ154は、ビデオゲーム記憶装置54のプリント回路基板70の端の電気接点74と接続する(図1参照)。このように、コネクタ154は、コプロセッサ200を記憶装置ROM76に電気的に接続する。更に、コネクタ154は、記憶装置の記憶装置セキュリティプロセッサ152をメインユニットのシリアル周辺インタフェイス138に接続する。特定の実施例では、コネクタ154は、主に書き込み不可能読取り専用メモリ7

6からデータおよび命令を読みとるが、メインユニット52は、コネクタが双方向性であるように、つまりメインユニットがビデオゲーム記憶装置54に情報を送り、かつビデオゲーム記憶装置54から情報を読みとることができるように設計されている。

【0042】図8は、コプロセッサ200のオーディオおよびビデオ出力が、カラーテレビ受像機58に送られる前に、コプロセッサ200外部の電子回路で処理されることを示している。特に、この実施例では、コプロセッサ200は、オーディオおよびビデオ出力をデジタル形式で出力するが、従来の家庭用カラーテレビ受像機58は、一般にアナログオーディオおよびビデオ出力を必要とする。そのため、コプロセッサ200のデジタル出力は、アナログ形式に変換される。この機能は、オーディオ情報についてはDAC140により、ビデオ情報についてはVDAC144により実行される。DAC140のアナログオーディオ出力は、メインユニット52の外部で発生され、コネクタ154を介して供給されたオーディオ信号を混合もするオーディオ増幅器142により増幅される。VDAC144のアナログビデオ出力は、ビデオエンコーダ146に供給される。ビデオエンコーダ146は、例えば「RGB」入力信号を複合ビデオ出力に変換する。オーディオ増幅器142の増幅されたステレオオーディオ出力とビデオエンコーダ146の複合ビデオ出力は、コネクタ（図示せず）を介して家庭用カラーテレビ受像機58に与えられる。

【0043】図8に示すように、メインメモリ300は、CPU命令108bの形式でビデオゲームプログラムを記憶する。これらのCPU命令108bは、通常はビデオゲーム記憶装置54からコピーされる。この実施例のCPU100は、記憶装置ROM76からの命令を直接実行できるが、ROMからの各命令にアクセスするのに必要とされる時間は、メインメモリ300からの各命令にアクセスするのに必要とされる時間よりも非常に大きい。そのため、メインプロセッサ100は、通常はROM76からのゲームプログラム／データ108aを、ブロックでの必要に応じてメインメモリ300にコピーし、命令を実際に実行するためにメインメモリをアクセスする（図9、ブロック160e、160f参照）。メインプロセッサ100は、好ましくは、更に命令アクセス時間を減少するために内部キャッシュメモリを有している。

【0044】図8は、ビデオゲーム記憶装置54が、特定のビデオゲームのグラフィックおよび音響を供給するために必要とされるグラフィックおよび音響データ112aのデータベースも、記憶していることを示している。メインプロセッサ100は、必要に応じてビデオゲーム記憶装置54からグラフィックスおよび音響データ112aを読み取り、これらのデータをテクスチャデータ116、音響データ112b、およびグラフィック

スデータ112cの形式で、メインメモリ300に記憶する。この実施例では、表示プロセッサ500は、その内部に必要に応じてテクスチャデータ116がコピーされる、内部テクスチャメモリ502を備えている。

【0045】ビデオゲーム記憶装置54は、コプロセッサマイクロコード156も記憶する。前述したように、この実施例では、信号プロセッサ400は、種々のグラフィックスおよびオーディオ機能を実行するために、コンピュータプログラムを実行する。このコンピュータプログラムつまり「マイクロコード」は、ビデオゲーム記憶装置54により供給される。マイクロコード156は、ビデオゲーム記憶装置54により与えられるので、異なる記憶装置は異なるマイクロコード与えることができ、これによりソフトウェアの制御下で、コプロセッサ200により供給される特定の機能を創り出す。一般的には、メインプロセッサ100は、信号プロセッサをスタートする時は、いつでもマイクロコード156の一部をメインメモリ300にコピーし、また、信号プロセッサ400は、必要に応じてマイクロコードの他の部分にアクセスする。信号プロセッサ400は、信号プロセッサ400内のインストラクションメモリ402のマイクロコードを実行する。SPマイクロコード156は、信号プロセッサの内部インストラクションメモリ402に一度に全部を入力するには大き過ぎるので、信号プロセッサ400が別のタスクを実行できるようにするため、異なるマイクロコードの部分は、メインメモリ300から内部インストラクションメモリ402にロードされることが必要とされる。例えば、SPマイクロコード156の一部は、グラフィックス処理のため信号プロセッサ400にロードされ、一方、SPマイクロコード156の他の部分は、オーディオ処理のため信号プロセッサにロードされる。この実施例では、信号プロセッサマイクロコードRAM402（および図8には示されていない付加的な信号プロセッサデータメモリRAM）は、メインプロセッサ100のアドレス空間にマッピングされ、その結果、メインプロセッサは、ソフトウェアの制御下で、ロード命令および書き込み命令に応じてRAMの内容に直接アクセスできる。

【0046】[メインプロセッサ100] この実施例における、メインプロセッサ100は、カリフォルニア州、マウンテン・ビュー（Mountain View）のMIPS Technologies, Inc.で設計されたMIPS R4300 RISC マイクロプロセッサである。このR4300プロセッサは、整数および浮動小数点演算用の64ビットレジスタファイル、16KBのインストラクションキャッシュ、8KBのライトバックデータキャッシュ、および仮想アドレス／物理アドレス計算用の32エントリTLBを有する実行ユニットを備えている。メインプロセッサ100は、32ビットアドレスによりカーネルモードでCPU命令

108を実行する。64ビットの整数動作は、このモードで使用できるが、性能を最大にするには32ビットの呼出し規則が望ましい。メインプロセッサ100についてのこれ以上の情報については、例えば、Heinrich, MIPSマイクロプロセッサR4000ユーザーズマニュアル(MIPS Technologies, Inc., 1994, 第2版)を参照されたい。

【0047】メインプロセッサ100は、バス102を介してコプロセッサ200と通信する。この実施例では、バス102は、双方向性32ビットSysAD多重 10 アドレス/データバス、双方向性5ビットワイドSysCMDバス、および更なる制御ライン、およびタイミングラインから構成されている。前述のHeinrichマニュアルの第12章、およびそれ以降を参照されたい。

【0048】従来のR4300メインプロセッサは、6つのハードウェア割込み、1つの内部(タイマー)割込み、2つのソフトウェア割込み、および1つの非マスク割込み(NMI)をサポートしている。この実施例では、この6つのハードウェア割込み入力のうちの3入力 20 (INT0、INT1およびINT2)と、この1つの非マスク割込み(NMI)の入力とによって、ビデオゲームシステム50の他の部分が、メインプロセッサに割り込むことが可能になる。詳細には、メインプロセッサの割込みINT0は、コプロセッサ200がメインプロセッサに割り込むことを可能にするために接続されており、メインプロセッサの割込みINT1は、ビデオゲーム記憶装置54がメインプロセッサに割り込むことを可能にするために接続されており、メインプロセッサの割込みINT2およびNMIは、シリアル周辺インタフェイス 30 138がメインプロセッサ100に割り込むことを可能にするために接続されている。メインプロセッサは、割込まれる時はいつも、その割込みの原因を決定するために内部割込みレジスタを参照し、次に、適切な方法で(例えば、状態レジスタを読みとって、あるいは他の適当な処理を実行して)これに応答する。ほとんどのシリアル周辺インタフェイス138からのNMI割込み入力は、マスクできる(つまり、メインプロセッサ100は、ソフトウェアの制御下で、それらの入力を選択的に使用可能および使用禁止にできる)。

【0049】メインプロセッサ100は、CPU-コプロセッサバス102を介して、ビデオゲームシステム50の残りの部分からデータを読み取り、そこへデータを書き込む。コプロセッサ200は、メモリマッピング機能を実行し、メインプロセッサ100が、メインメモリ 300、記憶装置カートリッジROM76、シリアル周辺インタフェイス138内の「ブートROM」150 (およびシリアル周辺インタフェイス138の他の部分)、コプロセッサ200のそれぞれの部分(信号プロセッサRAM402を含む) およびビデオゲームシステ 50

ム50の他の部分にアドレスできるようにする。

【0050】この実施例では、メインプロセッサ100により実行される動作は、完全にビデオゲームプログラム108に依存している。この実施例では、全ての「システム」ソフトウェアは、最大のフレキシビリティを与えるために記憶装置58から供給される。異なるビデオゲーム(あるいは他のアプリケーション)は、違う種類のハイレベルソフトウェアで、より効率的に実行できる。従って、本実施例におけるメインユニット52は、いかなる標準ソフトウェアライブラリ(あるいは、一切のソフトウェアライブラリ)をも提供しない。かかるソフトウェアライブラリは、フレキシビリティを制限する可能性があるからである。その代わりに、この実施例では、全てのソフトウェアがビデオゲーム記憶装置54により供給される。

【0051】ビデオゲームプログラム108の開発者は、ビデオゲームシステム50内の各種の資源を管理するために、新型ソフトウェアアーキテクチャ、例えば、デバイスドライバ、スケジューラ、およびスレッドライブラリを使用したいと考えるかもしれない。メインプロセッサ100は、最先端のRISCプロセッサ/コンピュータであるので、そのようなソフトウェアアーキテクチャ/構成を使用することおよびビデオゲームプログラム108をハイレベルのソフトウェア環境で実行することは適切である。

【0052】メインプロセッサ100のアドレス空間のシステム「メモリマップ」の実施例が、図10に示されている。図10に示されているように、メインメモリ300は、この実施例では2つのバンク(バンク0およびバンク1)に分割されている。さらに、メインメモリ300内の特定の構成のレジスタ307が、コプロセッサ200内のレジスタと同様に、メインプロセッサ100のアドレス空間にマッピングされる。この実施例では、メインプロセッサ100は、ビデオゲームプログラム108の制御下で、コプロセッサ200の各サブブロックに関連する制御レジスタに書き込むことにより、種々のコプロセッササブブロックのそれぞれを制御できる。

【0053】図10に示されているように、ビデオゲーム記憶装置54のアドレス空間は、(例えば、2つの異なるデバイスに対して)2つの「領域」に分割されている。これらの「領域」は、メインプロセッサ100のアドレス空間の幾つかの部分にマッピングされている。シリアル周辺インタフェイス138のそれぞれの部分(つまり、PIFブートROM150、PIFバッファRAM、およびPIF状態レジスタ)も、メインプロセッサ100のアドレス空間にマッピングされている。

【0054】[統合メインメモリ300]この実施例のメインメモリ300は、カリフォルニア州、マウンテン・ビューのRambus Inc. から入手できるRD RAMダイナミックランダムアクセスメモリから成って

いる。この実施例では、メインメモリ300は、8メガバイトまでの記憶容量を与えるように拡張できるが、コストを下げるために、メインユニット52は、小容量のRAM（例えば、2あるいは3MB）で出荷すべきである。

【0055】メインメモリ300は、この実施例では、ビデオゲームシステム50全体について記憶容量を与える。これは例えば（図8に示されたように）、以下のデータ構造を含む全ての重要なデータ構造を記憶する単一のアドレス空間（図10参照）を提供する。

- ・メインプロセッサ命令108
- ・信号プロセッサマイクロコード156
- ・表示リストグラフィックコマンド110a
- ・再生リストオーディオコマンド110b
- ・テクスチャマップ116および他のグラフィックデータ112c
- ・カラー画像フレームバッファ118a
- ・奥行（Z）バッファ118b
- ・音響データ112b
- ・オーディオ出力バッファ114
- ・メインプロセッサのワーキング値
- ・コプロセッサのワーキング値
- ・システムの各部間で通信されたデータ

【0056】ラスタスキャン表示システムに、シングルアドレス空間メモリアーキテクチャを使用する利点および欠点は、知られている（例えば、Foley他、"コンピュータ・グラフィックスの基礎と練習" 177~178頁（第2版 Addison-Wesley 1990）を参照）。これまで、多くのビデオゲーム（および他のグラフィックス）システムのアーキテクチャは、グラフィックスデータには専用ビデオRAMデバイスを使用し、また他の形式のデータには他の形式のメモリデバイスを使用し、シングルアドレス空間のアーキテクチャを拒絶していた。しかし、統合メインメモリ300は、ビデオゲームシステム50のこの特定の実施例で多数の利点を与える。例えば、以下のようである。

【0057】システム要素間のデータ通信が簡単化される。一度データがメインメモリ300に記憶されると、システムの他の部分とデータを通信する際の追加のオーバーヘッドは、少しだけ必要かあるいは全く必要でない。システムの異なる部分間でデータを転送するオーバーヘッドは、最小にされる。例えば、メインプロセッサ100およびコプロセッサ200内の各サブブロックは、それぞれシステムのメインメモリ300にアクセスできるので、データ構造を記憶するため全てのシステム要素により使用されるメインメモリ300は、要素間の汎用通信チャンネル／データバッファとしても使用できる。

【0058】例えば、メインプロセッサ100がメインメモリ300内に記憶した表示リスト110は、信号プ

ロセッサ400により直接アクセスされることができる。同様に、メインプロセッサ（および／または信号プロセッサ）がメインメモリ内に記憶した表示コマンドは、表示プロセッサ500により直接アクセスされることができる。メインプロセッサ100のワーキングデータ（「キャッシュフラッシュ」を介して、メインメモリ300に自動的に書き込まれる）は、システムの全ての他の部分にも直ちに利用できる。

【0059】統合メモリは、メモリ割当てのフレキシビリティを与える。メインメモリ300のロケーションは、似ているように見え、そのため各ロケーションは、いかなる形式のデータ構造を記憶するためにも使用できる。メインメモリ300の全ての割当ての決定は、アプリケーションのプログラマに委ねられている。これは、データ構造のサイズおよびメモリの使用の点で、大きなフレキシビリティを与える。データ構造は、メインメモリ300内の任意のところに記憶でき、メインメモリ300の各ロケーションは、アプリケーションのプログラマが特定した所にどの様にでも割り当てることができる。

【0060】例えば、あるビデオゲームプログラムは、高解像度画像および／または画像スクロールおよびパンニングのために大きいフレームバッファを提供し、一方、他のビデオゲームプログラムは、他のデータ構造（例えば、テクスチャあるいはオーディオデータ）に対してメモリ空間を解放するように、より小さいフレームバッファを使用することを決定できる。あるアプリケーションは、メインメモリ300の記憶容量の多くをオーディオデータ構造に、かつ少しをグラフィックデータに向け、一方、他のアプリケーションは、記憶容量の多くをグラフィック関連データに向けることができる。同じビデオゲームプログラム108は、異なる効果を実現するために、ゲームプレイのある部分から他の部分へ（例えば、ゲームがレベルを変更する時に）メモリ割り当てをダイナミックにシフトできる。アプリケーションのフレキシビリティは、固定したつまりハードワイヤドメモリ割当てによって制限されない。

【0061】統合RAMアーキテクチャは、フレキシブルデータ構造のシェアリングおよび使用をサポートする。重要なデータ構造は、全て共通のメインメモリ300内に記憶されているので、それらは全てメインプロセッサ100および他のシステム要素によりアクセスできる。表示画像とソース画像との間に、ハードウェアの区別はない。例えば、メインプロセッサ100は、もし必要ならば、フレームバッファ118内のそれぞれの画素を直接アクセスできる。表示プロセッサ500のスキャン変換出力は、テクスチャマッピング処理のためのテクスチャとして使用できる。画像ソースデータおよびスキャン変換された画像データが、特殊な効果を、例えばスキャン変換された画像を視点にワーピングすること

を実現するために交換でき、および/または組合せることができる。

【0062】統合メモリアーキテクチャ(例えば、システムの異なる部分によるメインメモリ300へのアクセスのための競合)の欠点は、注意深いシステム設計により最小にされた。メインメモリ300は、この実施例では、単一の狭い(9ビット幅)バス106を介してアクセスされるが、許容帯域幅は、バスを非常に高速に(例えば、240MHzのオーダーで)することによって実現される。データキャッシュは、各サブ要素に、メインメモリ300が使用可能になるのを待つ余裕をもっと持たせるように、ビデオゲームシステム50全体に備えられている。

【0063】[コプロセッサ200]図8は、コプロセッサ200が、信号プロセッサ400および表示プロセッサ500に加えて、幾つかの要素を備えていることを示している。すなわち、以下の要素である。

- ・CPUインタフェイス202
- ・シリアルインタフェイス204
- ・パラレル周辺インタフェイス206
- ・オーディオインタフェイス208
- ・ビデオインタフェイス210
- ・メインメモリのDRAMコントローラ/インタフェイス212
- ・メインインタフェイスバス214
- ・タイミングブロック216

この実施例では、メインバス214は、コプロセッサ200内の種々の主要な要素のそれぞれが、相互に通信することを可能にする。

【0064】図11は、コプロセッサ200のより詳細な図であり、コプロセッサ200は、全てが同時にアクティブになり、かつ並列に動作するというプロセッサ、メモリインタフェイス、および制御論理の集まりであることを示している。以下は、コプロセッサ200のこれらの他のサブブロックのそれぞれにより提供される全機能を、簡単に説明している。

- ・信号プロセッサ400は、オーディオおよびグラフィックスのタスクを実行するマイクロコード化エンジンである。
- ・表示プロセッサ500は、フレームバッファ118にレンダリングするグラフィックス表示パイプラインである。
- ・コプロセッサシリアルインタフェイス204は、この実施例では、シリアル周辺インタフェイス128とコプロセッサ200との間のインタフェイスを与える。
- ・コプロセッサパラレル周辺インタフェイス206は、ビデオゲーム記憶装置54と、あるいはコネクタ154に接続された他のパラレルデバイスとインタフェイスする。
- ・オーディオインタフェイス208は、メインメモリ3

00内のオーディオバッファ114から情報を読みとり、それをオーディオDAC140に出力する。

・コプロセッサビデオインタフェイス210は、メインメモリ300内のフレームバッファ118aから情報を読み取り、それをビデオDAC144に出力する。

・CPUインタフェイス202は、メインプロセッサ100、コプロセッサ200、およびビデオゲームシステム50の残りの部分の間のゲートである。

・DRAMコントローラ/インタフェイス212は、これを介してコプロセッサ200(およびメインプロセッサ100)がメインメモリ300にアクセスする通路である。メモリインタフェイス212は、メインプロセッサ100、信号プロセッサ400、表示プロセッサ500、ビデオインタフェイス210、オーディオインタフェイス208、およびシリアルおよびパラレルインタフェイス204、206に対して、メインメモリ300へのアクセスを与える。

種々のプロセッサおよびインタフェイスの各々は、同時にアクティブになる。

【0065】この実施例では、信号プロセッサ400は、前述のインストラクションメモリ402、データメモリ404、スカラ処理ユニット410、およびベクトル処理ユニット420を有している。インストラクションメモリ402は、スカラ処理ユニット410および/またはベクトル処理ユニット420による実行のためにマイクロコードを記憶している。データメモリ404は、スカラ処理ユニット410およびベクトル処理ユニット420に対する入力データ、作業データおよび出力データを記憶している。信号プロセッサ400は、この実施例では、インストラクションメモリ402からの命令だけを実行するが、ダイレクトメモリアccess(DMA)方式により、メインメモリ300へアクセスする。

【0066】この実施例では、スカラ処理ユニット410は、MIPS R4000の命令セットのサブセットを実行する汎用整数プロセッサである。これは、インストラクションメモリ402内のマイクロコードにより特定される汎用作業を実行するために使用される。ベクトル処理ユニット420は、数値計算を並列に実行できる8つの16ビット計算要素から構成されている。ベクトル処理ユニット420は、特にグラフィックスマトリックス計算およびある種のデジタルオーディオ信号処理作業に適している。

【0067】表示プロセッサ500は、この実施例では、表示画像のデジタル表示を行わせるグラフィックス表示パイプラインエンジンである。これは、信号プロセッサ400および/またはメインプロセッサ100により発生されたグラフィックス表示コマンドに基づいて動作する。表示プロセッサ500は、テクスチャメモリ502に加えて、ラスターライザ504、テクスチャユニット506、カラーコンバイナ508、ブレンダ51

0、およびメモリインタフェイス512を備えている。簡単には、ラスターライザ504は、表示画面60上のどの画素がこれらのプリミティブの範囲内にあるか決定するために、多角形（例えば、三角形および矩形）の幾何学的プリミティブをラスターライズする。テクスチャユニット506は、テクスチャメモリ502内に記憶されたテクスチャマップを、ラスターライザ504により解かれたプリミティブエッジ方程式により確定されたテクスチャ領域上に貼り付ける。カラーコンバイナ508は、テクスチャの色とグラフィックプリミティブに
10 関連する色とを結合し、その間を補間する。ブレンダ510は、得られた画素とフレームバッファ118内の画素（フレームバッファ内の画素は、メモリインタフェイス512を介してアクセスされる）とをブレンドし、Zバッファリング（つまり、隠面除去およびアンチエイリアシング動作）を実行することにも専念する。メモリインタフェイス512は、個々の画素に対する読み取り、修正および書き込み動作を実行し、またテクスチャメモリ502をロード／コピーし、矩形を修正し（高速クリ
20 ア）、そして多重画素をテクスチャメモリ502からフレームバッファ118へコピーするという特殊のモードを有している。メモリインタフェイス512は、メインメモリ300へのアクセス回数を減らすために、1以上の画素キャッシュを有している。

【0068】表示プロセッサ500は、表示プロセッサの状態を記憶する回路514を有している。この状態情報は、例えば、レンダリングモードを選択し、かつモード変更により行われる全ての先行のレンダリングがモード変更が実行される前に発生することを確実にするために、表示プロセッサ500の残りの部分により使用され
30 る。

【0069】表示プロセッサ500用のコマンドリストは、通常は、信号プロセッサを表示プロセッサに接続する専用「Xバス」218を介して、信号プロセッサ400から直接入来する。詳細には、この実施例では、Xバス218は、表示プロセッサ500による処理のために、グラフィックス表示コマンドを信号プロセッサメモリ404から表示プロセッサ500内のコマンドバッファ（図11には図示せず）に転送するために使用され
40 る。しかし、この実施例では、信号プロセッサ400および／またはメインプロセッサ100が、メインメモリ300を介してグラフィックス表示コマンドを表示プロセッサ500に供給することもできる。

【0070】表示プロセッサ500は、内部テクスチャメモリ502にロードし、ブレンドのためにフレームバッファ118を読み取り、奥行き比較のためにZバッファ118Bを読み取り、Zバッファおよびフレームバッファに書き込み、そして、メインメモリに記憶された任意のグラフィックス表示コマンドを読みとるために、物理アドレスを用いてメインメモリ300にアクセスす
50

る。

【0071】[コプロセッサ内部バスアーキテクチャ]
図12は、コプロセッサバス214の構成の実施例を示すより詳細な図である。この構成は、この実施例では、32ビットアドレス（「C」）バス214Cおよび64ビットデータ（「D」）バス214Dから成っている。これらのバス214C、214Dは、信号プロセッサ400、表示プロセッサ500、CPUインタフェイス202、オーディオインタフェイス208、ビデオインタフェイス210、シリアルインタフェイス204、パラレル周辺インタフェイス206、およびメインメモリ（RAM）インタフェイス212のそれぞれに接続されている。図12に示すように、メインプロセッサ100とコプロセッサ200の各サブブロックは、内部コプロセッサバス214C、214D、およびメインメモリインタフェイス／コントローラ212a／212bを介してメインメモリ300と通信する。

【0072】この実施例では、メインメモリインタフェイス／コントローラ212a／212bは、9ビットワイドメインメモリ多重アドレス／データバス106を介して通信するために、コプロセッサアドレスバス214C上に現れたメインメモリアドレスを9ビットワイドフォーマットに変換し、かつメインメモリバス106の9ビットワイドデータフォーマットとコプロセッサデータバス214Dの64ビットワイドデータフォーマットとの変換を行う。この実施例では、DRAMコントローラ／インタフェイス212は、その一部としてRambus Inc. により提供される従来のRAMコントローラ212b（図12参照）を備えている。9ビット幅のメインメモリバス106の使用により、コプロセッサ200のチップピン数が低減される。

【0073】この実施例では、各コプロセッサ200の図示のサブブロックは、独自にメインメモリ300にアドレスしアクセスすることを可能にする、関連のダイレクトメモリアクセス（DMA）回路を有している。例えば、シングルプロセッサDMA回路454、表示プロセッサDMA回路518、オーディオインタフェイスDMA回路1200、ビデオインタフェイスDMA回路900、シリアルインタフェイスDMA回路1300、およびパラレル周辺インタフェイスDMA回路1400のそれぞれは、関連のコプロセッササブブロックがコプロセッサアドレスバス214C上にアドレスを発生し、かつ、コプロセッサアドレスバス214Dを介してデータを通信することを可能にする（更に、表示プロセッサ500は、フレームバッファ118およびテクスチャデータ116にアクセスするために、別のメモリインタフェイスブロック512を有している）。

【0074】各コプロセッサ200のサブブロックは、独自にメインメモリ300にアクセスすることができるが、この実施例では、それらは共通バス214C、21

4Dを共用している。また、サブブロックの一つだけが、一度にそれらの共用バスを使用できる。従って、コプロセッサ200は、それらの共用バス214を最も効率的に使用するように設計されている。例えば、コプロセッサ200のサブブロックは、同じサブブロックによる異なるバスアクセス頻度を最小にし、かつ、サブブロックが一時的なバス使用不能に対してもっと余裕をもつように、情報をバッファあるいは「キャッシュ」することができる。専用バス218は、メインバス214が使用可能になるのを待たずに、信号プロセッサ400が表示プロセッサ500と通信できるようにする。

【0075】図12に示すように、コプロセッサ200の各サブブロックは、CPUインタフェイス202を介して、メインプロセッサ100によりアクセスできる制御/状態レジスタを含んでいる。例えば、信号プロセッサレジスタ407、表示プロセッサレジスタ507、オーディオインタフェイスレジスタ1207、ビデオインタフェイスレジスタ907、シリアルインタフェイスレジスタ1307、パラレル周辺インタフェイスレジスタ206、RAMインタフェイスレジスタ1007a、およびRAMコントローラレジスタ1007bは、それぞれメインプロセッサ100のアドレス空間にマッピングされる。メインプロセッサ100は、コプロセッサ200内のサブブロックの動作を直接制御するために、ビデオゲームプログラム108の制御下で、これらの種々のレジスタを読み取りおよび/またはそこに書き込むことができる。

【0076】[信号プロセッサ400]図13は、この実施例の信号プロセッサ400のアーキテクチャをもっと詳しく示している。前述したように、信号プロセッサ400は、スカラユニット410、ベクトルユニット420、インストラクションメモリ402、およびデータメモリ404を備えている。この実施例では、スカラユニット410は、MIPS4000命令セットのサブセットを実行する32ビット整数プロセッサである(MIPS4000のアーキテクチャの下で、スカラユニット410の「CP1」コプロセッサとして定義されている)。ベクトルユニット420は、8つの16ビットセットの値について並列に整数計算(例えば、乗算、加算、減算および複合/累積)を実行する。

【0077】ベクトルユニット420は、8対の16ビットオペランドについて並列に同時に同じ演算を実行できる。これにより、信号プロセッサ400が、特に「積の和」演算、例えば、マトリクス乗算、テクスチャリサンプリング、およびオーディオデジタル信号処理、例えば、デジタルオーディオ合成、および空間/周波数フィルタリングに適合するようになる。

【0078】信号プロセッサ400は、インストラクションメモリ402内に存在する命令に基づいて高性能マシン制御を行うために、RISC(リデュース・インス

トラクション・セット・コンピュータ)アーキテクチャを使用している。この実施例では、実行ユニットは、バス434を介してインストラクションメモリ402にアドレスするために使用されるプログラムカウンタ432を備えている。この実施例では、信号プロセッサ400により実行されるべき全ての命令が、まずインストラクションメモリ402内に配置されていることが必要とされるが、プログラムカウンタ432は、インストラクションメモリ402内の4キロバイト命令空間だけをアクセスできる。実行ユニット430は、現在実行されている特定の命令に基づいて出力制御信号436を発生する。出力制御信号436は、信号プロセッサ400の他の全てを制御し、パイプライン命令処理を管理するために順序づけされる。スカラユニット410およびベクトルユニット420は、これらの制御信号436によって制御される。例えば、スカラユニット410は、バス438を介してデータメモリ404にアドレスし、ロード/記憶ブロック440を使用してデータメモリ404からデータを読み取り、および/またはそこにデータを書き込む。データバス414は、演算の結果に応じてテストを行い、得られた状態出力をバス442を介して、実行ユニット430に与える。実行ユニット430は、これらの状態出力を使用し、条件付き分岐あるいは飛び越しを実行し、適切な(次の)アドレスを有するプログラムカウンタ432をインストラクションメモリ402にロードする。スカラユニット410は、このようなより幅広い能力を備えているので、スカラユニット410は、この実施例では、32ビットの整数演算に加えて、例えば制御フロー、アドレス計算等の汎用機能に使用される。

【0079】実行ユニット430は、標準MIPS R4000命令セットに従って、中間、飛び越しおよびレジスタ命令フォーマットを実行する。図14(a)は、レジスタ命令フォーマット450の一実施例を示し、また、どのように信号プロセッサ400が、このレジスタ命令フォーマットを用いてデータメモリ404内の3つの128ビットワイドワード452をアクセスするかを示している。レジスタ命令フォーマット450は、6ビット操作コードフィールド450(a)、5ビットソースレジスタ規制子450(b)、5ビットターゲット(ソース/行先)レジスタ規制子450(c)、5ビット行先レジスタ規制子450(d)、およびパラメータフィールド450(e)を有している。パラメータフィールド450(e)は、シフト量および/または機能を特定でき、更に操作コードフィールド450(a)は、実行されるべき演算を定義する。フィールド450(b)、450(c)、および450(d)のそれぞれは、データメモリ404内のロケーションを特定し、また、それぞれは128ビット語を指定する。

【0080】図14(b)に示されているように、ベク

10

20

30

40

50

トルユニット420は、128ビット語のそれぞれを8つの16ビット値の連結シーケンスとして扱い、16ビット値のそれぞれを並列に演算する。ベクトルユニット420の演算は、通常はMIPS R4000命令セット内に浮動小数点演算のために格納されているCPI形命令内の命令により、呼び出される(この実施例では、信号プロセッサ400は、浮動小数点ユニットを有していない)。

【0081】スカラユニット410は、32個のレジスタを内蔵するレジスタファイル412を有している。各レジスタは、32ビット幅である。スカラユニット410は、整数演算および他の演算を実行するために必要な加算回路、シフト回路および他の論理回路を含むデータバス414を有している。レジスタファイル412は、MIPS R4000アーキテクチャにより定義される汎用レジスタファイルと同様であり、R4000フォーマット内の命令を受け取る。データバス414は、整数乗算器/割算器を備え、インストラクションメモリ402から64ビット幅命令を受け取る実行ユニット430と共に動作する。

【0082】ベクトルユニット420は、8組のレジスタファイル422(0)~422(7)、および8組の対応データバス423(0)~423(7)を有している。データバス423は、それぞれ16ビット乗算器、16ビット加算器、および48ビット累算器を有している(48ビット累算は、オーディオフィルタを多数のタップに適合させ、また、16ビット精度以上を要求するグラフィックス演算に32ビット結果を得るために、一連の16ビット乗数および和が使用されている部分積にも適合する)。レジスタファイル422のそれぞれが、32個の32ビット幅のレジスタを有している。128ビット幅のデータバス444は、ベクトルユニット420をロード/記憶ブロック440に接続し、また別の128ビット幅のデータバス446は、ロード/記憶ブロック440をデータメモリ404に接続する。データメモリ404は、4096(4KB)語を記憶し、各語は、128ビット幅である。データメモリ404内の語がベクトルユニット420による使用のために検索される時は、それは8つの16ビットセグメントにスライスされる。各セグメントは、ベクトルユニット420内の別のレジスタファイル422に送られる(図14(b)参照)。図14(c)は、ベクトルユニット420により実行される加算演算の実施例を示している。ベクトルユニット420が、データメモリ404内でアドレス指定された行先に書き込んだ時に、レジスタファイル422のそれぞれが、データメモリに書き込まれる前に128ビット語に結合される16ビットとなる(図14

(a)参照)。また、ロード/記憶ブロック440は、データメモリ内の128ビット語のうち16ビットサブ語を、別のレジスタファイル422へ、または別のレジ

スタファイル422から仕分けるステアリングマルチプレクサ構造(図示せず)を備えている。この際、特定のサブ語および特定のベクトルユニットレジスタファイルは、インストラクションメモリ402からの命令に基づいて選択できる。同様に、ロード/記憶ブロック440は、データメモリ408とスカラユニット410との間で異なるサイズのデータユニット(例えば、バイト、16ビットハーフ語、あるいは32ビット語)を仕分ける別のステアリングマルチプレクサ構造(図示せず)を備えている。この際、特定のデータユニットおよびサイズは、インストラクションメモリ402内の命令によって特定できる。例えば、Heinrich, MIPS R4000マイクロプロセッサ・ユーザーズ・マニュアル(第2版、1994)における「バイト」、「ハーフワード」、「ワード」、「ワードレフト」、「ワードライト」のロードおよび記憶の記載を参照のこと。

【0083】信号プロセッサ400は、DMAコントローラ454およびCPU制御レジスタ456を有している。DMAコントローラ454は、コプロセッサ内部バス214に接続され、インストラクションメモリ402および/またはデータメモリ404へデータを転送するために、またはインストラクションメモリ402および/またはデータメモリ404からデータを転送されるために使用される。例えば、DMAコントローラ454は、マイクロコードモジュール156をメインメモリ300から信号プロセッサインストラクションメモリ402にコピーできる。DMAコントローラ454は、またデータメモリ404とメインメモリ300との間で情報を転送するためにも使用される。DMAコントローラ454は、実行ユニット430により命令され、バス438を介してスカラユニットデータバス414からDMAアドレスおよびデータ情報を受け取る。DMAコントローラ454は、またCPU制御レジスタ456を介してメインプロセッサ100により命令される。CPU制御レジスタ456は、メインプロセッサ100のアドレス空間にマッピングされ、MIPS「CP0」命令フォーマットを用いて、信号プロセッサ400および実行ユニット430によりアクセスされる。

【0084】図15(d)~図16(1)は、CPU制御レジスタ756の実施例を示している。図15(d)~図15(h)に示されたレジスタは、DMAコントローラ454を制御および/または監視するために使用される。

【0085】例えば、図15(d)に示されたSP-D RAM DMAアドレスレジスタ458は、メインプロセッサ100(SP実行ユニット430と同様)により書き込まれあるいは読み取られ、またインストラクションメモリ402あるいはデータメモリ404内のスタートDMAアドレスを特定するために使用される。図15(e)に示されたSPメモリDMAアドレス460は、

メインメモリ300内のスタートDMAアドレスを特定するために使用される。図15(f)および図15

(g)に示された読み取りおよび書き込みDMA長さレジスタ462、464は、それぞれ信号プロセッサ400とメインメモリ300との間で転送されるべきデータブロックの長さを特定する。転送の方向は、これら2つのレジスタのどちらの一方が、ブロック長を特定するために使用されるべきかに応じて決められる。図15

(h)~図15(i)に示されたDMA状態レジスタ466、468は、それぞれDMAコントローラ454がフル、つまりビジーであるか否か決定するために、メインプロセッサ100により読み取られる。

【0086】図16(j)は、CPU制御レジスタ456内の主SP状態レジスタ470を示している。SP状態レジスタ470は、メインプロセッサ100により書き込まれた時にSP制御レジスタとして作用し(図16(j)中の上図)、メインプロセッサ100により読み取られた時にSP状態を指示する(図16(j)中の下図)。状態レジスタとして使用された時に、SP状態レジスタ470は、SPが停止されるか(フィールド471)、SPが区切点モードで動作しているか(フィールド472)、DMAコントローラ454がビジー(フィールド474)あるいはフル(フィールド475)であるか、SPI/Oがフルであるか(フィールド476)、SPがシングルステップモードで動作しているか(フィールド477)、SPがブレイクポイントに到達する時に割り込みをしないというモードで動作しているか(フィールド478)、およびSPが種々のソフトウェア依存パラメータに関する状態を与えるために、ソフトウェアの制御下で定義できる種々の汎用「信号」479を発生したか、をメインプロセッサ100に知らせる。メインプロセッサ100は、レジスタ470に書き込むことにより、信号プロセッサ400を停止あるいはスタートし(フィールド480、481)、ブレイクポイントモードをクリアし(フィールド482)、割り込みモードをクリアあるいはセットし(フィールド483、484)、シングルステップモードをクリアあるいはセットし(フィールド485、486)、ブレイクポイントモードで割り込みをクリアあるいはセットし(フィールド487、488)、および種々のソフトウェア依存「信号」をクリアあるいはセットする(フィールド489、490)ことができる。

【0087】図16(k)は、メインプロセッサ100と信号プロセッサ400との間の汎用通信のために、「セマフォ」として使用される別のSPレジスタ491を示している。SPレジスタ491は、メインプロセッサ100がレジスタを読み取る際にセットし、レジスタに書き込む際にクリアするフラグを備えている。信号プロセッサ400は、このフラグもセットあるいはクリアできる。

【0088】図16(l)は、SP命令メモリのBIST状態レジスタ492を示している。BIST状態レジスタ492は、メインプロセッサ100により書き込まれる時はBIST制御レジスタとして使用され(図16(l)中の上図)、またメインプロセッサ100により読み取られる時にBIST状態を示す(図16(l)中の下図)。プログラムカウンタ432は、また、メインプロセッサ100によって書き込まれ、そして読み取ることができるように、好ましくは、CPU制御レジスタ456にマッピングされる。

【0089】[信号プロセッサのマイクロコード] 信号プロセッサ400が実行する特定の機能は、ビデオゲーム記憶装置54により与えられたSPマイクロコード156によって決まる。この実施例では、SPマイクロコード156は、グラフィックスおよびオーディオ処理機能の両方を与える。前述したように、グラフィックス処理に対して信号プロセッサ400により実行されるメインタスクは、表示リストの読み取り、3D幾何学変換とライティング演算の実行、および表示プロセッサ500で使用する対応グラフィックス表示コマンドの発生を含んでいる。より詳細には、信号プロセッサ400は、マイクロコード156の制御下で、以下の全グラフィックス機能を実行する。

- ・表示リストの処理
- ・マトリクスの定義
- ・頂点発生およびライティング
- ・テクスチャの定義/ローディング
- ・クリッピングおよびカリング
- ・表示プロセッサコマンドのセットアップ
- ・フロー制御

信号プロセッサ400は、オーディオを処理するためにマイクロコード156の制御下で、以下の全機能を実行する。

- ・再生リストの処理
- ・デジタルオーディオ合成/処理
- ・メインメモリアーディオバッファ114への、デジタルオーディオサンプルの書き込み

【0090】[タスクリスト] メインプロセッサ100は、信号プロセッサにタスクリストを与えることにより、何をなすべきかを信号プロセッサ400に知らせる。信号プロセッサ400上を走るマイクロコード156のプログラムは、タスクと呼ばれる。メインプロセッサ100(およびビデオゲーム記憶装置54により供給されるビデオゲームプログラム108)は、信号プロセッサ400上のタスクをスケジューリングし呼び出すことに対応できる。このタスクリストは、タスクを実行するために走るのに必要なマイクロコード156のルーチンについてのポインタを含み、信号プロセッサ400がタスクを実行するのに必要な全ての情報を含んでいる。メインプロセッサ100は、ビデオゲームプログラム1

08の制御下で、このタスクリストを供給する。

【0091】図17は、タスクリスト250の一実施例を示している。このタスクリスト250は、1または1以上の表示リストおよび/または再生リスト110を参照する。この表示リストおよび/または再生リスト110は、次に他の表示リストまたは再生リストを含む別のデータ構造を参照する。表示リスト110は、別の表示リストおよび/またはグラフィックデータを指定できる。同様に、再生リストは、別の再生リストおよび/または音響データを参照できる。この実施例では、表示リストおよび再生リストは、10レベルまでの深さの階層データ構造として考えることができる。信号プロセッサ400は、スタックの表示リストおよび再生リストを処理し、現在の表示リストのポインタをプッシュしポップする。全ての表示リストは、「エンド」コマンドで終了する。例えば、図17に示された表示リスト110

(1)は、他の表示リスト110(2)を参照する。表示リスト110(2)は、このリストを実行するために必要とされるグラフィックデータ112を参照する。同様に、図17に示された再生リスト110(4)は、音

響データ112bを参照する。
【0092】グラフィックスアニメーションに関しては、フレーム毎に変化する表示リスト110の部分だけを「ダブルバッファ」することが望ましい。このように、フレームの次に変化するデータだけが「ダブルバッファ」されることが必要であり、このようにしてメインメモリ300内に空間を確保できる。ダブルバッファ間のスワッピングは、タスクリスト250内のセグメントベースアドレスを変更することにより、および適切に効率的な方法で階層化表示リストを組織することにより、効果的に行われる。表示リストあるいは表示リストのフラグメントは、更に効率的なメモリ使用のために、一体に連結できる。

【0093】図18は、信号プロセッサ400により新しいタスクリストの処理を呼び出すために、メインプロセッサ100により実行される処理の実施例を示している。メインプロセッサ100は、まずタスク(表示)リストをメインメモリ300内にロードする(ブロック601)。メインプロセッサ100は、次にSP状態レジスタ470に書き込みおよび/またはそこから読み取ることにより、信号プロセッサ400を停止する(あるいは信号プロセッサが停止されたことを保証するためにチェックする)(ブロック602)。次に、メインプロセッサ100は、SPDMAレジスタ458、460、462に書き込み、初期マイクロコードモジュールを信号プロセッサのインストラクションメモリ402にロードする(ブロック604、図18)。次に、メインプロセッサ100は、ブロック601によりロードされたタスク(表示)リストのメインメモリ300内のアドレスを、信号プロセッサのデータメモリ404に記憶する

(ブロック606、図18)。メインプロセッサ100は、次に信号プロセッサのプログラムカウンタ432をリセットし(ブロック608、図18)、また信号プロセッサ400をスタートするために、これをSP状態レジスタ470に書き込む(ブロック610、図18)。次に、信号プロセッサ400は、通常はタスク(表示)リストをメインメモリ300からデータメモリ404に取り込むために、DMAコントローラ454を使用する。

【0094】タスクリストを備えている信号プロセッサ400がスタートされると、タスクリストで要求された各動作の実行を進める。信号プロセッサ400は、処理がタスクリストの終了に達するまでタスクリストを実行し続け、終了時では処理が停止されメインプロセッサ100が新しいタスクリストを与えるのを待つ。一般に、メインプロセッサ100は、各ビデオフレームについて一度だけ新しいタスクリストを与えるが、前述したように多くの場合に、タスクリストの一部および/またはタスクリストが参照する表示および/または再生リストだけは、実際にフレーム毎に変化する。メインメモリ300内のタスクリストの部分は、「ダブルバッファ」されており、そのためメインプロセッサ100は、あるバッファに書き込み、一方信号プロセッサ400は、別のバッファから読み取ることができる。次のビデオフレームの前に、メインプロセッサ100は、ポインタを変更して、信号プロセッサ400に新しいバッファへのアクセスを与える。

【0095】信号プロセッサ400がタスクリストを実行するとき、特定のタスクを実行する必要に応じて、メインメモリ300から別のSPマイクロコード156モジュールを検索する。例えば、信号プロセッサ400は、そのDMAファシリテイ454を使用して、特定のグラフィックマイクロコードをインストラクションメモリ402にロードし、タスクリストによって特定されたグラフィックコマンドを実行する。同様に、信号プロセッサ400は、オーディオ処理マイクロコードルーチンを検索しロードして、タスクリストによって特定されたオーディオ処理を実行する。異なるマイクロコードルーチンつまり「オーバーレイ」が必要に応じてロードされ、特定の形式のグラフィックスおよび/またはオーディオ処理動作を更に最適に処理する。一実施例として、信号プロセッサ400は、特別のライティンググラフィックスルーチンをオーバーレイとしてロードして特定のライティング動作を実行し、またクリッピングルーチンつまりオーバーレイをロードして特定のクリッピング動作を実行する。信号プロセッサのインストラクションメモリ402が、SPマイクロコード156の全部を記憶するのに十分な大きさを持っておらず、また、信号プロセッサ400は、その内部インストラクションメモリからの命令だけを実行できるように設計されているので、一つ

のタスクリスト250の実行中に、信号プロセッサ400にマイクロコードをローディングおよびリローディングすることが必要である。

【0096】図19は、表示リスト110に基づいて信号プロセッサ400により実行される、簡略化されたグラフィックス処理の実施例を示している。この簡略化された処理では、表示リスト110は、まず信号プロセッサ400に命じて、コプロセッサによりレンダリングされるべき全体グラフィックス画像を定義する種々の属性をセットする。これらの属性は、例えば、シェーディング、ライティング、Zバッファリング、テクスチャ発生、フォギング、およびカリングを含んでいる（図19、ブロック612）。表示リスト110は、次に、信号プロセッサ400に命じて、モデリング/ビューイングマトリクスおよび投影マトリクスを定義する（図19、ブロック614）。一度適切なマトリクスが定義されると、表示リスト110は、信号プロセッサ400に命じて、ブロック614により定義されたモデリング/ビューイングマトリクスおよび投影マトリクスに基づいて、またブロック612によりセットされた属性に基づいて、頂点のセットを変換する（図19、ブロック616）。最後に、表示リスト110は、信号プロセッサ400に命じて、ブロック616により発生された頂点とブロック612によりセットされた属性とに基づいて、プリミティブをレンダリングするように表示プロセッサ500に指示するグラフィック表示（例えば、三角形）コマンドを発生する（図19、ブロック618）。信号プロセッサ400は、ステップ618に応答して、表示プロセッサ500によるアクセスおよび実行のために発生された表示プロセッサコマンド（つまり、信号プロセッサ400がデータメモリ404あるいはメインメモリ*

*300内に記憶したコマンドのアドレス）を転送する。

【0097】図20は、表示リスト110を処理するために（例えば、図19に示された形式の処理を実行するために）、信号プロセッサ400のグラフィックマイクロコード156により実行される全体処理620を示している。信号プロセッサ400は、次の表示リストコマンドを入手し、それがどんな種類のコマンドか決定する（図20、ブロック622）。この実施例の表示リストコマンドは、一般に5つの異なる形式がある。

- 10 ・信号プロセッサの属性コマンド
- ・表示プロセッサコマンド
- ・マトリクスコマンド
- ・頂点コマンド
- ・三角形コマンド
- ・フロー制御コマンド

【0098】表示リストコマンドが信号プロセッサの属性コマンドであると、信号プロセッサ400は、コマンドにより特定されたように信号プロセッサの属性をセットする（図20、ブロック624）。この実施例では、

- 20 以下の形式のSP属性コマンドが定義されている。

- ・シェーディング
- ・ライティング
- ・Zバッファリング
- ・テクスチャリング
- ・フォギング
- ・カリング

以下は、SP属性コマンドフォーマットおよび関連の定義の実施例である。

【0099】[信号プロセッサの属性コマンド]

G_SETGEOMETRYMODE:

【表1】

コマンド		コマンド

このコマンドは、レンダリングパイプライン状態のあるものを「セット」する。この状態は、信号プロセッサ400内で保持され、セット/クリアインタフェイスがユーザに提供される。コマンドフィールド内で「オン」にあるビットは、内部状態でオンにされる。

G_SHADE

頂点シェーディングを可能にする、または多角形をベイントするプリミティブカラーを使用可能にする（デフォルト時は、頂点シェーディング）。

G_LIGHTING

ライティング計算を可能にする。

G_SHADING_SMOOTH

スムーズつまりフラットシェーディングを可能にする（デフォルト時は、このビットのクリアによりフラットシェーディング）。

G_ZBUFFER

Zバッファ実行計算を可能にする。

G_TEXTURE_GEN

テクスチャ座標SおよびTの自動発生を可能にする。変換後、球形マッピングが、当初与えられたSおよびT値を頂点に置き換えるために使用される。

- 40 G_FOG

発生されるべきフォグ係数を使用可能にし、頂点アルファを置き換える。アルファが大きければ、よりフォギーに（遠くへ）なる。

G_TEXTURE_GEN_LINEAR

G_TEXTURE_GENがセットされた時に発生されるテクスチャ座標の線形化を可能にする。例えば、これにより、環境マッピングの実行時にパノラマテクスチャマップが使用可能となる。

G_LOD

- 50 ミップマップされたテクスチャに細部レベル（LO

D) 値の発生を可能およびテクスチャエッジモードを可能にする。

G_CULL_FRONT

前向き多角形を選択する。

G_CULL_BACK

後向き多角形を選択する。

[0100] G_CLEARGEOMETRYMOD * [表2]

コマンド		パラメータ	長さ = 16
セグメント	アドレス		

↓

ライト r	ライト g	ライト b	0x00
ライト r	ライト g	ライト b	0x00
ライト x	ライト y	ライト z	0x00

このコマンドは、光をレンダリングパイプラインに送る。環境光に加えて7つまでの(1~7と番号付けされた)指向性光がある。パラメータは、この光記述に置き換えるべき光の番号(n)を特定する。8つの光のうち幾つを使用すべきか特定するために、G_NUM_LIGHTSコマンドを使用する。特定された光の数がNならば、N個の光(1~N)が使用されるべきであり、N+1番目の光が環境光である。「パラメータ」フィールドは、データメモリ404+(n-1)×2に保持されている値に基づいて、セットされるべきである。

[0102] 環境光は、色すなわち光r、光g、光b(符号なし8ビット整数)により定義され、これは描画対象の色により多重化されている環境光の色にセットされるべきである(テクスチャマッピングされたオブジェクトがライティングされる場合には、環境光の色が使用される)。(環境光としては、光x、光y、光zフィールドは無視される。)この実施例では、環境光は、黒色を特定する以外はオフにできない。

※

コマンド		パラメータ	長さ = 8
セグメント	アドレス		
↓			
0x8000		32 × (1+N)	
0x00000000			

N=拡散光源の数(1~7)

このコマンドは、幾つの光が使用されるべきか決定する。これは、常にG_LIGHTコマンドの後で、次のG_VTXコマンドの前に使用されるべきである。このパラメータは、少なくとも1以上7以下の拡散光源の数☆

40☆(N)を特定する。環境光源は、光数N+1であり、指向性光源は、1からNに番号付けされた光である。

[0106] G_SETOTHERMODE_H:

[表4]

コマンド	シフト	長さ
ワード		

このコマンドは、ブレンディング、テクスチャリング、およびフレームバッファのパラメータを含む表示プロセッサ500内の「他の」モードのハイワードをセットする。信号プロセッサ400は、簡単なセットコマン

*E: G_SETGEOMETRYMODEと同じであるが、このコマンドは、レンダリングパイプライン状態のあるものの幾つかを「クリア」する(コマンドフィールド内で「オン」にあるビットが、内部状態でオフにされる)。

[0101] G_LIGHT:

[表2]

※ [0103] 指向性光は、色すなわち光r、光g、光b(符号なし8ビット整数)により定義され、これは環境光と同様に描画対象の色により多重化されている光源の色にセットされるべきである。指向性光も方向を持っている。光x、光y、光zフィールド(7ビット小数を有する符号あり8ビット小数)は、照明するオブジェクトからの方向を示している。(G_LIGHTTINGがG_SETGEOMETRYMODEコマンドで使用可能にされている場合には)少なくとも一つのオンにされた指向性光がある。しかし、その色が黒である場合には、シーンには影響がない。

[0104] 光の数が変更されない場合でも、G_NUM_LIGHTSコマンドは、GLIGHTコマンドの後で、次のG_VTXコマンドの前に、常に使用されるべきである。

30 [0105] G_NUM_LIGHT:

[表3]

ドインタフェイスを示すために、表示プロセッサ500内の「他の」状態のハイおよびローワードを記憶している。このコマンドは、表示プロセッサのコマンドであるが、信号プロセッサ400により解析され解釈される。

そのため、このコマンドは、信号プロセッサを通さずに直接に表示プロセッサに送ることは、まずできない。

【0107】このコマンド内のシフトおよび長さパラメータは、以下のマスクを構成するために使用される。

$((0 \times 01 \ll \text{長さ}) - 1) \ll \text{シフト}$

このマスクは、表示プロセッサ500の状態ワード内のこれらのビットをクリアするために使用される。ワードパラメータからの新しいビットは、状態ワードにORさ*

コマンド	s スケール		
t スケール	ミッドマップレベル	タイル数	オン

このコマンドは、テクスチャマッピングをオン/オフし、テクスチャ座標スケールを与え、(張られるテクスチャ内の)タイル数を選択する。スケールパラメータは、(16)のフォーマットであり、頂点コマンド内のテクスチャパラメータを拡大縮小する。テクスチャのオン/オフは、幾何学パイプラインのテクス*

コマンド	パラメータ	長さ = 16
セグメント	アドレス	



0x00000000			
0x00000000			
X	Y	Z	0x00

このコマンドは、テクスチャ座標の自動発生に使用される。このコマンドは、信号プロセッサ400がテクスチャ座標を発生すべきものについて知ることができるように、目の方向性を記述する。XYZ値(7ビット小数を有する8ビット符号あり小数)は、ワールド空間(モデルビューマトリクスとプロジェクションマトリクスとの間の空間)内のベクトルを記述している。この空間は、視聴者の視る方向に垂直であり、視聴者の右側に向いている。

【0111】G_LOOKAT_Y: G_LOOKAT_Xと同じであるが、アドレス指定されたセグメント内の最初のゼロワードは、ゼロである(0x00000000)。

【0112】[DPコマンドの発生] 図20を再び参照すると、次の表示リストコマンドが表示プロセッサ500に向けたものである場合には、信号プロセッサ400は、単にコマンドを表示プロセッサに書き込む(図20のブロック626)。ブロック626は、表示プロセッサコマンドをXバス218を介して、表示プロセッサ500にダイレクトメモリアクセスするか、あるいは表示プロセッサによるアクセスのために、表示プロセッサコマンドをメインメモリ300内のバッファにセットする。

【0113】[マトリクスコマンド] 次の表示リストコマンドがマトリクスコマンドである場合には、信号プロ

される(パラメータワードは、予備シフトされる)。

【0108】G_SETOTHERMODE_L: G_SETOTHERMODE_Hと同じであるが、表示プロセッサ500の「他の」モードのローワードに作用する。

【0109】G_TEXTURE:

【表5】

※チャ座標処理をオン/オフする。タイル数は、パイプラインのラスタ部で選択されたタイルに相当する。タイル数は、細部レベル(LOD)についての最大レベルを保持する(ミッド・マッピング)。

【0110】G_LOOKAT_X:

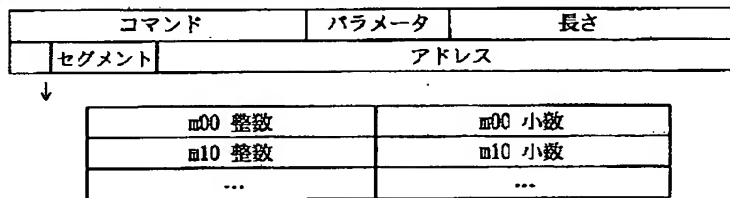
【表6】

セッサ400は、使用している現在のマトリクスの状態を更新し(図20、ブロック628)、更新したマトリクスをマトリクススタックに配置する(ブロック630)。前述したように、この実施例では、信号プロセッサ400は、10個の深いモデリング/ビューイングマトリクススタックを維持する。新しいマトリクスは、スタック上にロードでき、スタックの上部で乗算され(連結され)、あるいはスタックからポップオフされる。この実施例では、信号プロセッサ400は、「1つの深い」投影マトリクスを保持している。そのため、新しいマトリクスは、現在のマトリクスにロードできあるいは乗算できるが、プッシュあるいはポップできない。

【0114】この実施例では、モデリング/ビューイングマトリクススタックは、メインメモリ300内に存在する。ビデオゲームプログラム108は、このスタックに十分なメモリを割り当て、ポインタをタスクリスト250内のスタック領域に与えなければならない。マトリクスのフォーマットは、信号プロセッサのベクトルユニット420に最適化される。適切な解像度を提供するために、この実施例では、信号プロセッサ400は、各マトリクス値を32ビット「倍精度」で表す。上位16ビットは、符号あり整数部分(1より大きい値の部分を示す)に割り当てられ、下位16ビットは、小数部分(0と1との間の値の部分を示す)に割り当てられている。しかし、この実施例のベクトルユニット420は、16

ビットワイド値で動作し、直接に32ビットワイド値を乗算できない。マトリクスフォーマット(図21(b)示されたもの)は、要素の整数部分の全てをグループ化し、続いて要素の小数部分の全てをグループ化する。これにより、信号プロセッサ400は、マトリクスを繰り返し「アンパック」あるいは「パック」せずに、16ビットの整数部分と16ビットの小数部分とを別々に乗算することにより、マトリクスをより効率的に処理できる。

【0115】例えば、ベクトルユニット420は、マトリクス行の16ビット固定小数点符号あり整数値の各々をある演算で乗算でき、同じ行の16ビット小数部分の各々を別の演算で乗算できる。これらの2つの部分の結*



このマトリクスコマンドは、パラメータフィールドのフラグにより制御される方法で、引き続き幾何学を変換するために使用される4×4変換マトリクス(図21(b)参照)を指定する。長さは、入来するマトリクスバイトのサイズである。このコマンドにより指定された4×4変換マトリクスは、以下のフォーマットを有している。それは、メモリの連続したブロックであり、ROW MAJORの順でマトリクスの16の要素を含んでいる。マトリクスの各要素は、固定小数点フォーマットS15.16の形式である。バイトでの4×4マトリクスの長さは、64バイトである。セグメントidおよびアドレスフィールドは、メインメモリ300の実際のマトリクスのアドレスを構成するために使用される(もっと情報が必要な場合には、G_SEGMENTS Pを参照のこと)。

【0117】パラメータフィールドでは、以下のフラグが使用される。

G_MTX_MODELVIEW

入来マトリクスを、シェーディング等のための多角形法線の効率的な変換を行うのに必要とされるモデルビューマトリクスとして識別する。(デフォルト時)

G_MTX_PROJECTION

入来マトリクスを、シェーディング等のための多角形法線の変換に影響しない投影マトリクスとして識別する。

G_MTX_MUL

入来マトリクスは、マトリクススタックの現在の上部で連結される。(デフォルト時)

G_MTX_LOAD

入来マトリクスは、(モデルビューあるいは投影)マト

* 果は、一体にされ32ビット倍精度値を得るか、あるいは2つの部分の結果は(例えば、結果の整数部分だけあるいは結果の小数部分だけを要求する演算に対して)、別個に使用される。この実施例では、ベクトルユニット420が16ビット値で動作し、また明白な「倍精度」能力を持っていない場合でも、マトリクス表示は、信号プロセッサ400が32ビット精度値を効率的に処理することを可能する。以下は、信号プロセッサマトリクスコマンドおよび関連のフォーマットの例である。

【0116】[マトリクスコマンドの例]

G_MTX :

[表7]

リクススタックの現在の上部を置き換える。

G_MTX_NOPUSH

マトリクススタックの現在の上部は、スタックの上部でロードあるいは連結動作を実行する前にはプッシュされない。(デフォルト時)

G_MTX_PUSH

マトリクススタックの現在の上部は、スタックの上部でロードあるいは連結動作を実行する前にプッシュされる。プッシュは、G_MTX_MODELVIEWで支持されるだけで、G_MTX_PROJECTIONで支持されない。これは、投影マトリクススタック(投影は明白に再ロードされねばならない)が無いためである。

【0118】パラメータの組み合わせを有するこの単一のコマンドは、各種の通常に使用されるマトリクス演算を可能にする。例えば、(G_MTX_LOAD | G_MTX_NOPUSH)は、スタックの上部を置き換える。(G_MTX_MUL | G_MTX_PUSH)は、連結を実行し、スタックを一般的なモデリング階層構造にする。

【0119】ライティングおよびテクスチャリングに対しては、多角形の法線は、モデルビューマトリクスの逆転置により変換されねばならない(「オープンGLプログラミング・ガイド」参照)。これは、別個のモデルビューおよび投影スタックが維持され、入来マトリクスが識別されねばならないためである。

【0120】G_POPMTX :

[表8]

91	コマンド	92
		パラメータ

このコマンドは、モデルビューマトリックスのスタックをポップする。パラメータフィールドは0である。空のスタックは、ポップしない。投影マトリックスのスタックがないので、このコマンドは、モデルビューマトリックスに*

* 対してだけ支持される。

【0121】G_VIEWPORT:
【表9】

コマンド	パラメータ	長さ = 16
セグメント	アドレス	

x スケール	y スケール
z スケール	パッド
x 翻訳	y 翻訳
z 翻訳	パッド

このコマンドは、ビューポート構造をグラフィックスパイプラインに送る。セグメントidおよびアドレスフィールドは、実際のビューポート構造のメインメモリ300のアドレスを構成するために使用される（もっと情報が必要な場合には、G_SEGMENTを参照のこと）。

【0122】ビューポート変換は、正規化スクリーン座標のスケール変換である。一般に、ビューポートは、スクリーンデバイス座標のハードウェアの要件に適合する※

(SCREEN_WD/2*4, (SCREEN_HT/2) * 4, G_MAXZ, 0, /* scale */

(SCREEN_WD/2*4, (SCREEN_HT/2) * 4, 0, 0, /* translate */

※ために、投影マトリックスと共同で構成されねばならない。xおよびyに対するスケールおよび変換の項は、ハードウェア内のサブ画素の位置決めに適合するのに必要な2ビットの小数を有している。z値は、小数がない。【0123】省略時の投影マトリックスの一つを用いて小数ビットをカウントすることによって、ビューポート構造は、このように初期設定できる。

【数1】

【0124】【頂点コマンドの処理】図20を再び参照して、次の表示リストコマンドが「頂点コマンド」であれば、信号プロセッサ400は、現マトリクス状態において、この頂点コマンドにより特定され現ライティング状態により可能な限りシェードされた頂点を交換し、頂点についてクリッピングテストを実行し、得られた頂点をデータメモリ404内の408にロードする。この実施例では、信号プロセッサ400は、16個までの頂点を保持する頂点バッファを備えている。図22(a)は、メインプロセッサ100およびビデオゲームプログラム108に十分に露呈された信号プロセッサ400の頂点バッファを示している。16ポイントまでを保持できるこの内部頂点バッファ408は、信号プロセッサのデータメモリ404に記憶され、メインプロセッサ100により読み取られる。

【0125】この実施例の信号プロセッサ400は、線、三角形あるいは矩形（つまり、2、3、あるいは4頂点により定義されるサーフェイス）だけを扱うことができるが、この実施例の頂点バッファ408は、16個までの頂点を記憶する。そのため、信号プロセッサ400は、頂点をその度に再計算する代わりに、交換された頂点値を再使用できる。この実施例では、ビデオゲームプログラム108を創り出すために使用される3D許可

／モデリングソフトウェアは、頂点の再使用（およびスピード性能）を最大にするために、表示リスト110を最適に組織する。

【0126】図22(b)は、頂点バッファ408に記憶された各頂点を表すために、信号プロセッサ400が使用する頂点データ構造の実施例を示している。この実施例では、頂点に対応する交換されたx、y、zおよびwの値は、倍精度形式で記憶され、整数部分に続いて小数部分が存在する（フィールド408(1)(a)～408(1)(h)）。頂点カラー（r、g、b、α）がフィールド408(1)(i)～408(1)(l)に記憶され、頂点テクスチャ座標（s、t）がフィールド408(1)(m)、408(1)(n)に記憶される。更に、この実施例から、スクリーン空間の座標の頂点値（つまり、交換されそしてビューイング平面上に投影されたもの）は、フィールド408(1)(o)～408(1)(t)に記憶される（1/w値が倍精度形式で記憶されている）。スクリーン座標は、頂点により定義された多角形を描くために、表示プロセッサ500により使用される。交換された3次元座標は、クリッピングテストのために頂点バッファ408に保持される。多角形（頂点でない）はクリップされ、かつ頂点バッファ408内の頂点は多重多角形に対して再使用されるの

で、これらの変換された3D頂点値は、実行されるべき乗算可能なクリッピングのために記憶される。更に、頂点データ構造408(1)は、例えば、信号プロセッサ400がクリップテスト結果(つまり、頂点が6つの異なるクリップ平面の各々の内側にあるか外側にあるか)を特定するために使用できるフラグ408(1)(v)を有している。フィールド408(1)(s)、408(1)(t)に記憶された透視的投影係数は、表示プロセ

コマンド	n	v0	長さ
セグメント	アドレス		

↓

x		y	
z		フラグ	
s		t	
r または nx	g または ny	b または nz	a

。
。
。

このコマンドは、(N+1)ポイントを頂点バッファ内のロケーションv0から始まるベクトルバッファにロードする。セグメントidおよびアドレスフィールドは、メインメモリ300のアドレスの実際のVTX構造を構成するために使用される(さらに情報が必要な場合は、G_SEGMENT参照のこと)。全16頂点が4ビットで表現されることを可能にするために、頂点の数nは、「マイナス1」として符号化される。長さは、ポイントの数の16倍であり、(バイトでの)VTX構造のサイズである。頂点座標は、16ビット整数であり、テクスチャ座標sおよびtは、S10.5である。フラグは、この実施例では無視される。頂点は、(シェーディングのための)色あるいは法線を有している。色は、8ビット符号なし整数である。法線は、8ビット符号あり小数(7ビットの小数)である。(0×7fは+1.0にマップされ、0×81は-1.0にマップされ、そして0×0は0.0にマップされる)。法線ベクトルは、正規化されねばならない、つまり、 $\sqrt{(x^2 + y^2 + z^2)} \leq 127$

【0128】頂点コマンドを受け取る際に、信号プロセッサ400は、現在のモデリング/ビューイングマトリクスを用いて頂点コマンドで特定された頂点を変換する(図20、ブロック632)。Neider他、オープンGLプログラミング・ガイド(シリコン・グラフィックス 1993)の第3章(「ビューイング」)を参照のこと。これらの変換は、選択された視点に比例して、3次元空間内の頂点により表されるオブジェクトを配向させる。例えば、これらの変換は、選択した視点に比例して、表示されているオブジェクトを平行移動し、回転し、および/または拡大縮小する。このような変換計算は、8つの並列計算を同時に実行するために、信号プロ

*セッサのテクスチャ座標ユニット(以下に説明する)により実行される透視的座標動作のために保持されている。

【0127】以下は、幾つかの点を有する内部頂点バッファにロードするために使用される頂点コマンドフォーマットの一実施例である。

G_VTX:

【表10】

セッサのベクトルユニット420およびその能力に負担のかかる使用を強いる。この実施例では、変換された結果は、倍精度形式で頂点データ構造フィールド408(1)(a)~408(1)(h)に記憶される。

【0129】[クリップテスト] 信号プロセッサ400は、次にクリップテスト(図20、ブロック636)を実行し、変換された頂点がシーンの内側にあるか外側にあるかを決定する。6つのクリッピング平面が、ビューイングボリュームのサイドおよびエンドを定義する。各変換された頂点は、これらの6つの平面のそれぞれと比較され、比較の結果(すなわち、頂点がクリップ平面のどのサイドに位置するか)は、頂点バッファの「フラグ」フィールド408(v)に記憶される(図22(b)参照)。これらの結果は、「三角コマンド」に応答してクリッピングブロック646により使用される(以下を参照)。この実施例は、多角形をクリップし頂点をクリップしないので、図20のブロック636は、実際にはクリッピングを実行せず、単にクリップ平面に対して頂点位置をテストするだけであることが注目点である。

【0130】[投影] 信号プロセッサ400は、次に投影マトリクスを用いて頂点値を変換する(図20、ブロック638)。投影変換の目的は、2つの方法で使用されるビューイングボリュームを定義することである。ビューイングボリュームとは、オブジェクトがどのように2次元ビューイングスクリーン上に投影されるかを決定することである(つまり、遠近法あるいは正射法を用いることにより)。(オープンGLプログラミング・ガイドの90頁以降を参照。)(投影マトリクスが遠近法を定義している場合には)短縮特性をもって、あるいは(投影マトリクスが正射法を定義している場合には)正

射的に変換された結果の頂点は、3次元空間から2次元ビューイング平面上に投影されている。これらの座標値は、フィールド408(1)(o)~408(1)(t)において、頂点バッファデータ構造に書き込まれる(「1/w」値は、後の射影修正のために保持される)。

【0131】[ライティング] 信号プロセッサ400は、次に、頂点コマンドで特定された各頂点を「照明」するために、ライティング演算を実行する。ビデオゲームシステム50は、環境(均一)光、拡散(指向性)光、および(テクスチャマッピングを用いた)鏡面ハイライトを含む、多数の複雑なリアルタイム照明効果をサポートしている。この実施例でのライティング演算実行によると、信号プロセッサ400は、ライティング演算を実行するために、まず、SPマイクロコード108のオーバーレイをロードする。G_SETGEOMETRYMODEコマンドは、ライティング演算が使用可能にされたことを特定しなければならず、光は、前述のG_NUM_LIGHTSコマンドにより定義されねばならない。ライティング演算を実行するマイクロコード108の部分は、通常、信号プロセッサ400内には存在しないが、ライティングコールが行われた時は、オーバーレイを介して信号プロセッサ400に入力される。これは、あるオブジェクトが照明され、他のオブジェクトが静的に色づけされた状態における、レンダリングシーンに対する性能に密接な関連を有している。この実施例では、ライティングオーバーレイは、照明されたシーンのクリップされたオブジェクトを最小にする、あるいは完全になくすのに最も適した最高の性能を実現するために、クリッピングマイクロコードをオーバーライトする。

【0132】オブジェクトをライティングするために、オブジェクトを構成する頂点は、特定された色に代えて法線であればならない。この実施例では、法線は、その法線のx、y、z成分を表す3つの符号あり8ビット数により構成される(前述のG_VTXコマンドフォーマット参照)。各成分は、この実施例では、-128から+127の値の範囲にある。x成分は、頂点の赤色の位置に相当し、y成分は緑に、z成分は青に対応する。アルファは、変化なしのままである。前述したように、法線ベクトルは、正規化されねばならない。

【0133】オブジェクトが向きを変えた時にオブジェクトが現れる方法を変更することにより、ライティングは、奥行の効果も実現する。この実施例では、信号プロセッサ400は、シーン内の7個までの拡散光をサポートする。各光は、方向と色を有している。オブジェクトおよび視者の向きに関わらず、光の方向が変更されるまで、各光は(オープン「ワールド」に対して)同じ方向で照らし続ける。さらに、1つの環境光は、均一の照明を出す。陰影は、この実施例では、信号プロセッサ40

0では簡単にはサポートされない。

【0134】前述したように、ライティング情報は、光データ構造で信号プロセッサ400に送られる。拡散光の数は、0から7に変わることができる。赤、緑、青の値の変数は、光の色を表し、0から255の範囲の値をとる。添字x、y、zを有する変数は、光の方向を表している。方向は、光を示すことが決められている。このことは、光の方向は、光に対する方向を示し、光が照らしている方向を示さないことを意味する(例えば、光がワールドの左上方から入って来る場合には、方向は、x=-141、y=-141、z=0となる)。環境光を排除するために、プログラマーは、環境光が黒(0、0、0)であることを特定しなければならない。

【0135】G_lightコマンドは、表示リスト上の光のセットをアクティブにするために使用される。ひとたび光が活性化されると、これらの光は、次のセットの光が活性化されるまでとどまっている。これは、新しい光構造のセットアップは、信号プロセッサ400内の古い光構造をオーバーライトすることを意味する。光が効果を発揮するように光の計算をオンにするには、ライティングモードビットが、G_SETGEOMETRYMODEコマンドを用いてオンにされることが必要である。前述のライティング構造は、頂点バッファフィールド408(1)(i)~408(1)(l)に記憶する色の値を提供するために使用される。

【0136】[テクスチャ座標スケーリング/生成] 信号プロセッサ400は、次にテクスチャ座標スケーリングおよび/または生成を実行する(図20、ブロック642)。この実施例では、ブロック642により実行される演算は、鏡面ハイライト、反射マッピング、および環境マッピングを実現するものである。これらの効果を得るために、この実施例では、コプロセッサ200は、光あるいは環境の画像のテクスチャマップを使用し、視点からサーフェイス法線に対する角度に基づいて、テクスチャ座標s、tを計算する。このテクスチャマッピング法により、鏡面ライティングを実現するのに、各画素でサーフェイス法線を計算する必要性をなくすることができる。各画素でサーフェイス法線を計算することは、この実施例におけるビデオゲームシステム50にとっては計算的にあまりにも厳しいものである。

【0137】ほとんどの光からの鏡面ハイライトは、強度分布を示す指数関数あるいはガウス関数により、丸ドットを定義するテクスチャマップにより表される。シーンが蛍光灯あるいは光輝く剣等の他の特別に成形された光によるハイライトを含んでいる場合には、ハイライトのテクスチャマップが実現できるレンダリングにおける困難さはない。

【0138】この実施例では、表示プロセッサ500がテクスチャマッピング動作を実行するが、信号プロセッサ400は、これらの効果が要求された時に、各頂点

についてテクスチャ座標変換を実行する。信号プロセッサ400のテクスチャ座標変換の作動あるいは不動作は、G_SETGEOMETRYMODEコマンド内の値により特定される（前述の記載を参照）。さらに、G_SETGEOMETRYMODEコマンドは、例えば、環境マッピングを実行する時に、パノラマテクスチャマップの使用を可能にするために、発生されたテクスチャ座標の線形化を指定する。

【0139】この実施例では、信号プロセッサ400のテクスチャ座標の発生では、テクスチャを参照するため、sおよびt指標をそれぞれ導出するために、スクリーン空間のxおよびy方向の頂点法線の投影が行われる。各頂点における視点とサーフェイス法線との間の角度は、s、tを発生するために使用される。法線投影は、この実施例では、実際のs、t値を実現するために拡大縮小される。信号プロセッサ400は、視点の「後ろ」の頂点を0にマッピングし、正の投影をスケール値にマッピングする。

【0140】この実施例では、テクスチャリングは、信号プロセッサ400の属性コマンドのところで前述したG_TEXTUREコマンドを用いて作動される。このコマンドは、他のものの中から前述のテクスチャ座標マッピングを実行するスケール値を与える。

【0141】前述したように、信号プロセッサ400により実行されたテクスチャ座標マッピングは、この実施例においては、頂点のサーフェイス法線と視点との間の角度が計算できるように、視点の向きを特定する情報を必要とする。G_LOOKAT_XおよびG_LOOKAT_Yコマンドは、信号プロセッサ400により実行されるテクスチャ座標の自動発生における視点の向きを与える。変換されたテクスチャ座標値は、計算された場合には、信号プロセッサ400により頂点データ構造のフィールド408(1)(m)、408(1)(n)に記憶される。これらのテクスチャ座標値は、*

コマンド				
	N	v0	v1	v2

このコマンドは、内部頂点バッファに記憶されている頂点v0、v1およびv2を用いて、一つの三角形を生じさせる。Nフィールドは、3つの頂点のどれが（フラットシェーディングのための）面の法線、あるいは（フラットシェーディングのための）面の色を含んでいるか識別する。

コマンド				
	N	v0	v1	

このコマンドは、内部頂点バッファに記憶されている頂点v0およびv1を用いて、一本の線を生じさせる。Nフィールドは、2つの頂点のどれが（フラットシェーディングのための）面の色を含んでいるか特定する。

* G_TEXTUREコマンドにより特定されたテクスチャを用いて、得られたテクスチャマッピングを実行するために、表示プロセッサ500に供給される。

【0142】これらの効果は、テクスチャマッピングを使用するので、別の方法でテクスチャマッピングされるオブジェクトには使用できない。

【0143】[頂点バッファの書き込み]これらのステップの全てを実行した後に、信号プロセッサ400は、変換され、照明され、投影された頂点値を頂点バッファ408に書き込み（図20、ブロック644）、次の表示リストコマンドを解析することに戻る（ブロック622）。

【0144】[三角コマンド処理]一度、信号プロセッサ400が頂点を頂点バッファ408に書き込むと、表示リスト110は、「三角コマンド」を与えることができる。この「三角コマンド」は、頂点バッファ408内の頂点により定義された多角形を特定する。この「三角コマンド」は、基本的には、多角形を表すグラフィック表示コマンドを発生し、かつ、レンダリングのために、このコマンドを表示プロセッサ500に送るための信号プロセッサ400への要求である。この実施例では、信号プロセッサ400は、3つの異なる種類のプリミティブ、つまり、線、三角形、および矩形を与える。この実施例では、線あるいは三角形を与えるために、マイクロコード108の異なるモジュールがロードされることが必要である。この実施例では、全ての矩形は、スクリーン座標で特定された2次元プリミティブであって、クリッピングもシザリングもされない。

【0145】以下は、三角コマンドのフォーマットおよび関連機能の一実施例である。

【三角コマンドの例】以下のコマンドは、頂点バッファ内の3頂点により定義された三角形を特定する。

G_TRI1:

【表11】

※【0146】以下のコマンドは、信号プロセッサ400を制御して、頂点バッファ408内の2頂点により定義された線をレンダリングする表示プロセッサ500のコマンドを発生するために使用される。

【0147】G_LINE3D:

※【表12】

【0148】テクスチャされ、かつ満たされた矩形は、信号プロセッサ400の介在を要求し、それゆえに信号プロセッサの動作となる。以下は、テクスチャ矩形コマンドのコマンドフォーマットおよび関連の機能の

実施例である。

*【表13】

【0149】G_TEXRECT:

*

コマンド	x0	y0
	x1	y1

【表14】

コマンド	0x000000
S(左上方テクスチャ座標)	T(左上方テクスチャ座標)

【表15】

10

コマンド	0x000000
DsDx	DtDy

これらの3つのコマンドは、現在のテクスチャを有する2D矩形を描く。パラメータx0、y0は、矩形の左上方の隅を特定し、パラメータx1、y1は、右下方の隅を特定する。全座標は、12ビットである。SおよびTは、符号あり10.5ビット数であり、s、tの左上方の座標を特定する。DsDxおよびDtDyは、符号あり5.10ビット数であり、x(y)座標の変化に対するs(t)の変化を特定する。

【0150】この実施例において、信号プロセッサ400は、s座標がy方向に変化し、またはt座標がx方向に変化するように、テクスチャが動かされることを除き、G_TEXRECTコマンドと同じであるようなG_TEXRECT_FLIPコマンドをサポートする。

【0151】G_FILLRECT:

【表16】

コマンド	x0	y0
	x1	y1

このコマンドは、現在のフィルカラーで2D矩形を描く。パラメータx0、y0は、矩形の左上方の隅を特定し、パラメータx1、y1は右下方の隅を特定する。全座標は、12ビットである。

【0152】[クリッピング/セットアップ]図20に戻ると、信号プロセッサ400は、三角コマンドを受け取ると、頂点の必要なクリッピングを実行する(図20、ブロック646)。このクリッピング動作は、ビューイング平面を確定する6つのクリップ平面の外側に存在する幾何学プリミティブの部分を除く。

【0153】前述したように、各頂点について実行されたクリップテスト636の結果は、頂点バッファ408に記憶され利用できる。この三角コマンドが、これらの頂点により確定されたプリミティブを示している場合は、信号プロセッサ400は、このプリミティブのクリッピングを進める。プリミティブの全ての頂点が、6つのクリップ平面により確定された空間内に存在する場合には、全プリミティブは、表示空間内に存在し、クリッピングは必要ない。(図22(b)に示された頂点デー

タ構造408(1)のフラグフィールドにより示されたように)プリミティブを確定する全ての頂点が、同じクリップ平面の外側に存在する場合には、全プリミティブは、表示から排除され捨てられる。プリミティブを確定する頂点のあるものが表示空間内に存在し、一方あるものが表示空間の外側に存在する場合(あるいは、全ての頂点が表示空間の外側に存在するが、この表示空間を通過するプリミティブを確定する場合)には、プリミティブは、クリッピングされることが必要であり、新しい頂点が定義される。これらのテストおよび演算は、この実施例では、クリッピングブロック646により実行される。

【0154】信号プロセッサ400は、次に裏面カリリングを実行する(図20、ブロック647)。この演算は、オブジェクトの裏面上にあると決定され、視野から隠れている多角形を捨てることにより、描画速度を最大にする。この実施例では、フロントフェーシングであれ、バックフェーシングであれ、ブロック647により、どちらのプリミティブもカリリングされる(つまり、捨てられる)。カリリングすべきプリミティブのタイプは、前述のG_SETGEOMETRYMODEコマンド内のパラメータにより特定される。これにより、幾何学が任意の方向に並べられ、あるいは種々の効果を達成する異なるカリリングフラグが、使用されるところ(例えば、内部サーフェース、2辺多角形、等)に並べられる。

【0155】信号プロセッサ400は、セットアップ動作を実行し(図20、ブロック648)、また、グラフィック表示コマンドを表示プロセッサ500に送り、表示プロセッサ500がプリミティブをレンダリングするように制御する(図20、ブロック650)。セットアップ動作の一部として(ブロック648)、この実施例では、信号プロセッサ400は、表示リスト110内の「セグメント化」されたアドレスを、表示プロセッサ500が使用する物理アドレスに変換する(表示プロセッサ500は、この実施例では、物理アドレスマシンである)。

20

30

40

50

101

【0156】この実施例では、信号プロセッサ400は、メインメモリ300をアドレス指定する際に支援するようにセグメントテーブル416（図23参照）を使用する。より詳細には、信号プロセッサ400内のアドレスは、テーブル入力417aおよび26ビットオフセット417bにより示される。テーブル入力417aは、セグメントアドレステーブル416内の16のベースアドレスのうちの1つを参照する。参照されたベースアドレスは、オフセット417bに加えられ、メインメモ

コマンド	
セグメント	アドレス

このコマンドは、入力を前述のセグメントテーブル416に加える。信号プロセッサ400により使用されるセグメント化アドレス指定は、この実施例では、ダブルバッファされたアニメーションを容易化するのに有効である。例えば、ビデオゲームプログラム108は、2つの異なるセグメントに同じオフセットを持たせた状態で、メインメモリ300内のある表示リストのフラグメントの2つのコピーを保持することができる。それらのコピーを切り換えることは、信号プロセッサ400内のセグメントポインタを交換することと同じ程度に簡単である。別の使用法としては、あるセグメント内にデータとテクスチャをグループ化し、別のセグメント内に静的バックグラウンド幾何学をグループ化することがある。データをグループ化することは、メインプロセッサ100内のメモリキャッシングを最適化する助けとなる。埋め込みアドレスを含む全てのデータは、信号プロセッサ400のセグメントテーブルに正しいベースアドレスをロードする適当なG_SEGMENTコマンドにより先行されねばならない。

【0158】信号プロセッサ400は、図23に示されたセグメントアドレス指定法を使用しているが、この実施例では、この構成は、表示プロセッサ500には利用できない。そのため、セットアップ処理648の一部は、レンダリングに必要とされるデータ構造を指定するセグメントアドレスを、表示プロセッサ500により直接使用できる物理アドレスに変換せねばならない。

【0159】[DPコマンド書き込み]グラフィックスのための信号プロセッサ400の主な出力は、図20、ブロック650により出力される表示プロセッサ500への1以上のコマンドである。メインプロセッサ100（あるいはビデオゲーム記憶装置54）は、直接に表示プロセッサ500のコマンドを供給できるが、3D画像

102

* モリ300への物理アドレスを発生する。信号プロセッサ400は、セグメントおよび（例えば、表示リスト110により与えられる）26ビットオフセットにベースアドレスを加えることにより、メインメモリ300のアドレスを構成する。セグメントテーブル416は、以下の例のG_SEGMENTコマンドに基づいて構成される。

【0157】G_SEGMENT：
【表17】

に対しては、信号プロセッサ400は、変換され、投影され、照明され、クリッピングされ、カリングされたプリミティブを表現する表示プロセッサコマンドを発生するために、前述の変換処理を実行することが一般に必要とされる。

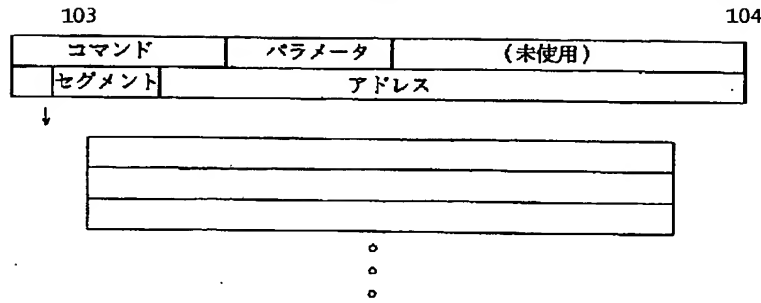
【0160】表示プロセッサコマンドのレパートリは、図65から図102に示す。信号プロセッサ400は、発生される表示プロセッサコマンドを適切にフォーマットすること、およびコマンド内に適切な情報およびアドレス情報を含ませることに対応する。さらに、信号プロセッサ400は、表示プロセッサが適切なパラメータを用いて、信号プロセッサ400により特定された特定のプリミティブをレンダリングするのに必要とされる適切なモードおよび属性コマンドを発生し、かつ出力できる（通常は、表示プロセッサ500へのモードおよび属性コマンドの多くが、ビデオゲームプログラム108の制御下で、メインプロセッサ100により直接供給されるが）。前述したように、メインプロセッサ100は、直接に表示プロセッサ500を提供できるが、3Dオブジェクトが変形されるべきときは、いつでも少なくともある表示プロセッサコマンドを発生するために、一般には、信号プロセッサに頼ることが必要である。

【0161】[フロー制御コマンド処理]再び図20に戻って、信号プロセッサ400から受け取った表示リストコマンドがフロー制御コマンドである場合には、信号プロセッサ400は、適切な方法でこのコマンドに応答して、表示リスト110を利用する、つまり、詳しく参照する。コマンドおよびフォーマットの以下の実施例は、フロー制御を与える。

【0162】[フロー制御コマンドの例]

G_DL：

【表18】



このコマンドは、別の表示リストを指定し、また表示リストの階層、入れ子表示リスト、間接参照等を生成するために使用される。セグメントフィールドは、メモリセグメントを識別する。アドレスフィールドは、セグメントのベースからのオフセットである。また、これらは新しい表示リストを指定するメインメモリ300内のアドレスを形成する。この実施例では、全ての表示リストが、G_ENDDLコマンドにより終了されることが望ましいが、長さフィールド(図示せず)は、新しい表示リストの長さをバイトで記述できる。パラメータフィー*

コマンド	

エンド表示リストコマンドは、表示リストの階層の分岐を終了し、表示リストの階層の処理において「ポップ」を生じさせる。このコマンドは、演繹的に表示リストの長さを与える代わりに、エンドコマンドで終了されるという変化するつまり未知サイズの表示リストピースを構※

コマンド			

このコマンドは、何も設定されていない。このコマンドは、ある状況下で内部的に発生される。

【0165】図20、ブロック652は、メインメモリ300内の表示リストのスタックを維持し、この表示リストのスタックをプッシングし、ヌーピング（横断）する機能を実行する。ブロック652は、信号プロセッサ400が「オープンエンド」表示リストコマンドに出会うと、信号プロセッサ400を停止する。

【０１６６】〔信号プロセッサのマイクロコードオーディオ処理〕この実施例の信号プロセッサ４００は、前述したグラフィックス処理に加えてデジタルオーディオ処理を実行する。信号プロセッサ４００のベクトルユニット４２０は、「積の和」計算を実行するのに特に適している。この「積の和」計算は、例えば、オーディオ圧縮解除、ウェーブテーブル・リサンプリング、合成、およびフィルタリング、等のオーディオ信号のある種のデジタル信号処理において特に有効である。かなり数多くのタップを用いたデジタル空間および／または周波数フィルタリングは、ベクトルユニットデータバス４２３に４８ビット幅の累算器が含まれていることから、精度を損

10* ルドは、転送の動態を制御するフラグを保持している。フラグG_DL_NOPUSHがセットされている場合には、現在の表示リストは、転送制御の前にはスタック上にブッシュされない。これは階層化表示リストよりも、もっとBRANCHあるいはGOTO命令のような投割を果たす（これは、より大きい表示リストを不連続メモリピースに分解し、次に、それらを表示リストの分岐により接続するのに有効である）。

【0163】G_ENDDL:
【表19】

※成するために、最も効果的である。全ての表示リストは、このコマンドで終了する。

【0164】G_NOOP:
【表20】

なわずに適応できる。オーディオ処理用のベクトルユニット420の特定の最適な使用法の一つの実施例として、信号プロセッサ400のベクトルユニット420の8個の別個のレジスタファイル422および関連データバス423は、8つの異なるMTDIオーディオを同時に並列に処理するために使用できる。以下は、ベクトルユニット420を用いて効果的に実行できるさらなるオーディオ処理の例である。

- ・多項式を解くことができる
- ・8つのオーディオつまり8倍のサンプルを並列に処理することができる
- ・立体補間を用いたウェーブテーブル合成、ここでは、ベクトルユニット420のデータバス423のうち4つが、1つのサンプルを処理するために使用され、他の4つのベクトルユニット420のデータバス423は、2つめのサンプルを処理するために使用される
- ・オーディオ包絡線処理、ここでは、8つのベクトルユニット420のデータバス423は、それぞれ異なるオーディオサンプルに異なる重み係数を乗算することができる

・オーディオミキシング処理、ここでは、8つのベクトルユニット420のデータバス423は、それぞれ異なるオーディオサンプルに対応するミキサ重み係数を乗算することができる

【0167】信号プロセッサ400は、オーディオデジタル信号処理を高速で効果的に実行できるので、オーディオ再生実時間間隔に関連したデジタルオーディオ処理を実行し、かつ完了するには、当該オーディオ再生実時間間隔の何分の1かが、かかる。例えば、信号プロセッサ400は、オーディオをデジタルに処理するには、1

／30秒よりずっと少ない時間しか必要とせず、コプロセッサのオーディオインタフェイス208は、これを第二の時間間隔の1／30で実時間で再生する。この能力の故に、この実施例では、信号プロセッサ400は、グラフィックス処理とデジタルオーディオ処理との間で時分割されている。

【0168】一般に、メインプロセッサ100は、ビデオフレームの始めて信号プロセッサ400にタスクリスト250を与える。このビデオフレームは、次に続くビデオフレーム中に発生されるべき画像および音響を特定

する。当該次に続くビデオフレームが始まるまでに、コプロセッサ200は、当該次に続くビデオフレームのためのオーディオとグラフィックス処理の両方を完了しなければならない。ビデオ表示およびオーディオ再生は、実時間の連続処理であるので（つまり、新しいビデオ表示は、各ビデオフレーム時間に供給されねばならず、またオーディオは連続的に供給されねばならない）、コプロセッサ200は、次のフレームが始まるまでに、各次に続くビデオフレームに関連する全てのオーディオおよびビデオ信号処理を終了する必要がある。

【0169】この実施例では、信号プロセッサ400は、デジタルオーディオ信号処理とグラフィックス処理とで共用されている。信号プロセッサのベクトルユニット420の高速計算能力の故に、信号プロセッサ400は、現ビデオフレーム時間よりもずっと短い時間で、次に続くビデオフレームの間に再生されるべきオーディオの処理を完了でき、また、現ビデオフレーム時間よりも短い時間で、次に続く画像の間に表示されるべき画像のグラフィックス処理も完了できる。これにより、タスクリスト250が、次のビデオフレーム時間が始まるまでに、信号プロセッサ400およびコプロセッサ200により完了されねばならないグラフィックス表示リストおよびオーディオ再生リストの両方を特定できる。しかし、この実施例では、メインプロセッサ100がコプロセッサ200に、コプロセッサ200が次のビデオフレーム時間が始まる前に、完了できないタスクリスト250を与えることを妨げるものは何もない。信号プロセッサ400により要求される複合グラフィックスおよびオーディオ処理が、十分に集中しており時間がかかるものであれば、信号プロセッサ400は、現ビデオフレーム

時間全体についてタスクリストの処理を続けることができ、また、次のビデオフレームが始まるまでには実行されない。ビデオゲームプログラム108は、コプロセッサ200を酷使しないようにし、もし酷使されてる場合は、適切な方法で過度の負担を処理すべきである。ビデオゲームプログラムは、全ての表示リスト110が効率的に組織されるようにし、3Dでオブジェクトを効果的な方法でモデリングし、広範囲に渡り時間のかかる処理（例えば、クリッピング）を避け、あるいは最小限にするように配慮することにより、信号プロセッサ400の過度の負担を避けることができる。しかし、このような配慮によっても、コプロセッサ200は、特に複雑な画像の処理を完了するには、1つのビデオフレーム時間以上の時間を必要とする。ビデオゲームプログラムは、コプロセッサ200が次の画像の処理を完了できる時間内の多重ビデオフレームの間に、フレームバッファ118の一部分に記憶される同じ画像を、テレビ受像機58が再表示するように実効フレーム速度を遅くすることにより、この状況を解決できる。ユーザは、望ましくない遅延として可変のフレーム速度を知覚する可能性があるので、最も処理が集中する画像を完了するには、全体の実効フレーム速度をコプロセッサ200のために、より要求される速度にまで低減することがしばしば最善である（これにより、より複雑な画像がより複雑でない画像よりもゆっくりと現れることを防ぐことができる）。

【0170】オーディオ処理に関しては、ユーザは、他の連続したオーディオのストリームにおいて邪魔な「クリック」音を聞くことになるので、所定のビデオフレーム時間の間にオーディオを供給し損なうことは、一般的に容認できないことである。このようなオーディオの妨害は、耳に入り易くうるさいものである。そのため、それらは避けるべきである。信号プロセッサ400が、割り当てられたオーディオ処理を時間内に完了できなかった状況下で耳につくオーディオ妨害を避ける1つの方法は、メインプロセッサ100が、オーディオインタフェイス208に次に続くフレームの間に、前のフレームに相当するオーディオを再生するように命令することである。注意深く行えば、ユーザが妨害に気づかずに、好ましいオーディオをこのように発生できる。他の方法は、

信号プロセッサ400に、1つのビデオフレーム時間内に、複数のビデオフレームに相当するオーディオを処理させることである。これにより、実効ビデオフレーム速度とは異なる（より早い）実効オーディオ「フレーム」速度を供給することができる。「実効フレーム速度」は、コプロセッサ200が、1つのフレームに相当する情報を発生させる速度を意味する（この例では、テレビ受像機の実際のビデオフレーム速度は、一定である）。

【0171】[オーディオソフトウェアアーキテクチャの実施例] 図24は、オーディオを合成し処理するためのビデオゲームシステム50により与えられるソフトウ

エアのアーキテクチャ全体の一実施例を示している。この全体ソフトウェアのアーキテクチャ700は、4つのソフトウェア対象、この実施例においては、シーケンスプレーヤ702、音響プレーヤ704、合成ドライバ706、およびオーディオ合成マイクロコード708を有している。この実施例では、シーケンスプレーヤ702、音響プレーヤ704、および合成ドライバ706は、全てメインプロセッサ100上で実行され、一方、オーディオ合成マイクロコード708は、コプロセッサの信号プロセッサ400上を走る。このように、シーケンスプレーヤ702、音響プレーヤ704、および合成ドライバ706は、それぞれ記憶装置54のゲームプログラム108の一部分として供給され、一方、オーディオ合成マイクロコード708は、SPマイクロコード156の一部分として供給される。

【0172】シーケンスプレーヤ702、音響プレーヤ704、および合成ドライバ706は、再生されている特定のビデオゲームによって異なる。一般に、シーケンスプレーヤ702は、タイプ0のMIDI音楽シーケンスファイルを再生する。シーケンスプレーヤ702は、シーケンス、インストゥルメントバンクおよびシンセサイザ資源の割り当て、シーケンス解釈、およびMIDIメッセージスケジューリングを取り扱う。音響プレーヤ704は、全てのADPCM圧縮オーディオサンプルを再生する。音響プレーヤ704は、音響効果および他のストリームオーディオに有効である。合成ドライバ706は、メインプロセッサ100によってソフトウェア制御の下でタスクにパッケージされ、かつコプロセッサ200にタスクリスト250の形で送られるオーディオ再生リスト110を生成する。この実施例では、合成ドライバ706は、音響プレーヤ704あるいは他の「クライアント」がウェーブテーブルをシンセサイザオーディオに割り当てること、および再生パラメータを制御することを可能にする。前述したように、オーディオ合成マイクロコード708は、送られたタスクを処理し、かつL/Rステレオ16ビットサンプルを合成する。信号プロセッサ400は、L/Rステレオ16ビットサンプルをオーディオインタフェイス208、オーディオDAC140、および増幅器/ミキサ142を介して、再生のためにメインメモリ300内のオーディオバッファ114に保管する。

【0173】この実施例では、合成ドライバ706は、オーディオタスクを信号プロセッサ400にオーディオ「フレーム」の形で送る。「フレーム」は、多数のオーディオサンプルであり、通常は、一定のビデオフレーム速度（例えば、30あるいは60Hz）で全ビデオフレーム時間を満たすのに必要なサンプルの数に近いものである。テレビ受像機58は、いかなるビデオフレーム速度パラメータ（例えば、テレビ受像機は、水平および垂直ビデオブランキングおよび帰線周期の間にオーディオ

を発生できる）によっても拘束されない連続したストリーム内で、オーディオ信号を受信し処理する。しかし、この実施例では、実行されるグラフィック関連のタスクがビデオフレーム速度に拘束されていることから、信号プロセッサ400（これは、オーディオおよびグラフィックス処理で共用されている）は、ビデオフレーム速度に従って動作しなければならないので、ビデオゲームシステム50は、ビデオフレーム速度に従ってオーディオ処理を構成している。

【0174】[再生リストの処理の実施例] 図25は、簡単な信号プロセッサの再生リストの処理の一実施例を示している。図25の処理は、ビデオゲームプログラム108の制御下で、メインプロセッサ100により生成された再生リスト110によって特定され、かつタスクリスト250の部分として特定される。このように、図25のSP再生リスト処理は、オーディオ再生リスト110の形で信号プロセッサ400に供給される合成ドライバ706の出力の一実施例を示すものである。

【0175】この実施例では、インストラクションメモリ402の大きさが制限されているため、一般にオーディオ合成マイクロコード708は、信号プロセッサ400内に連続的には存在しない。代わりに、メインプロセッサ100の初期設定マイクロコードは、インストラクションメモリ402にロードされるように構成され（図18、ブロック604参照）、また適切なオーディオマイクロコードルーチンが、オーディオ処理用のインストラクションメモリにロードされるようにする（また、適切なグラフィックスマイクロコードルーチンが、グラフィックス処理用のインストラクションメモリにロードされるようにする）。図25に示されたステップは、オーディオ合成マイクロコード708が、信号プロセッサのインストラクションメモリ402内に存在し、かつ、信号プロセッサ400は、示されたステップを特定するオーディオ再生リスト110を読み取ることを仮定している。

【0176】一般に、オーディオ再生リスト110の最初のタスクは、オーディオ処理タスクを実行するために必要とされる信号プロセッサのデータメモリ408内にバッファをセットアップすることである（図25、ブロック710）。一般に、バッファのセットアッププロセスは、1またはそれ以上のオーディオ入力バッファとして使用されるデータメモリ404内に領域を割り当てること、およびデータメモリ404内にオーディオ出力バッファを割り当てることを含んでいる。一般に、メインプロセッサ100は、信号プロセッサ400のDMAファシリティ454を使用するように信号プロセッサ400に命令し、メインメモリからオーディオ入力データ112bを取り出し、処理のために上記割り当てた入力バッファに入力する。メインプロセッサ100は、次に、オーディオ処理のために使用される一定の属性（例え

ば、ボリューム範囲および変化速度)をセットできる
(図25、ブロック712)。メインプロセッサ100は、次に、適切なパラメータとともに、信号プロセッサ400により実行される信号処理のタイプを特定する
(図25、ブロック714)。この実施例では、メインプロセッサ100は、個別にあるいは組み合わせて実行される圧縮解除、リサンプリング、包絡線/パン、ミキシング、および他の処理(例えば、リヴァーブ)を特定できる。オーディオ再生リスト110は、通常、信号プロセッサのデータメモリ404に記憶された出力オーディオバッファの内容をメインメモリ300に保存する
10 コマンドにより終了する(ブロック716)。

【0177】[オーディオ合成マイクロコードの実施例] 図26は、オーディオ合成マイクロコード708が、この実施例において実行する全タスクを示す。マイクロコードの制御下で、信号プロセッサ400は、現在のオーディオ再生リスト110から次の再生リストコマンドを取り出し、コマンドの種類が何であるかを決定する(図26、ブロック718)。この実施例では、オーディオ再生リスト110内のオーディオコマンドは、以下
20 の一般的タイプに該当する。

- ・バッファコマンド
- ・フロー制御コマンド
- ・属性コマンド
- ・圧縮解除コマンド
- ・リサンプリングコマンド
- ・包絡線/パンコマンド
- ・ミックスコマンド
- ・特殊信号処理/効果コマンド

【0178】[バッファコマンドの処理] バッファコマンドは、信号プロセッサのデータメモリ404内のオーディオバッファを管理し、オーディオデータのデータメモリとメインメモリ300の間での転送を可能にする。
30 以下は、バッファコマンドフォーマットおよび関連機能の実施例である。

【0179】[バッファコマンドの例]

A_SETBUFF:

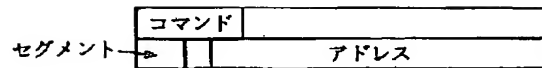
【表21】

コマンド	データメモリ入力
データメモリ出力	カウント

このコマンドは、内部の信号プロセッサのデータメモリ404のバッファポインタおよび処理コマンドが使用するカウント値を設定する。通常は、このコマンドは、あらゆる処理コマンドの前に発生される。「データメモリ入力」は、入力バッファを、「データメモリ出力」は、出力バッファを示し、「カウント」は、処理すべき16
ビットサンプルの数を定義する。

【0180】A_LOADBUFF:

【表22】



このコマンドは、「セグメント」+「アドレス」フィールドが与えたメインメモリ300のアドレスから、信号プロセッサのデータメモリ404のバッファをロードする。SPデータメモリのバッファローケーションおよびロードすべき16ビットサンプルの数は、A_LOADBUFFコマンドの前に、A_SETBUFFコマンドを
10 発生することにより定義される。

【0181】A_CLEABUFF:

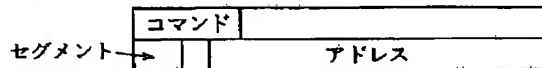
【表23】

コマンド	データメモリ入力
	カウント

このコマンドは、「データメモリ」が与えた信号プロセッサ400のデータメモリアドレスでスタートする、16ビットサンプルのサイズ「カウント」の領域をクリアする。

【0182】A_SAVEBUFF:

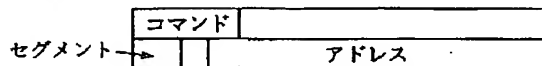
【表24】



このコマンドは、信号プロセッサのデータメモリ404にある16ビットサンプルのバッファを、「セグメント」+「アドレス」フィールドにより与えられたメインメモリ300のアドレスに保存する。入力SPデータメモリバッファおよびサンプルの数は、A_SETBUFF
30 コマンドを発生することにより定義される。

【0183】A_SEGMENT:

【表25】



グラフィックスのG_SEGMENTコマンドを参照。このコマンドは、間接「セグメント」アドレスを、メインメモリ300の物理アドレスにマッピングするために
40 使用される。

【0184】図26を再び参照すると、信号プロセッサのオーディオ合成マイクロコード708は、データを関連データメモリバッファ409に設定し、管理し、書き込み、またはメモリバッファ409からデータを読み出すことにより、特定されたバッファコマンドを実行する(図26、ブロック720)。通常は、信号プロセッサ400は、そのDMAファシリテイ454を使用して、データをメインメモリ300および信号プロセッサのメモリ404の間で転送し、処理するオーディオ入力データを取り出し、またはオーディオインタフェイス208
50 によって再生するオーディオデータをメインメモリ300

0に保存する。

【0185】[フロー制御コマンド処理] 次の再生リストコマンドが、フロー制御コマンドの場合は、信号プロセッサ400は、現在のオーディオ再生リストをコマンドで特定された方法で検索して、コマンドに応答する。オーディオ再生リスト110のネステイニングは、好ましくは許され、信号プロセッサ400は、オーディオ再生リストのスタックを、メインメモリ300に（グラフィックス表示リストを保持するのと同様に）保持することができる。

【0186】[属性コマンド処理] 次のオーディオ再生リストコマンドが属性コマンドの場合は、信号プロセッサ400は、その後のオーディオ処理に使用される適切なモード、および／または属性条件を設定することによりコマンドを処理する（図26、ブロック724）。この実施例において、オーディオ合成マイクロコード708は、下記の実施例の属性コマンドフォーマットおよび関連機能をサポートしている。

【0187】[属性コマンドの例]

A_SETVOL:

【表26】

コマンド	音量
音量目標	音量速度

このコマンドは、その後の処理コマンドに対して、ボリュームパラメータをセットするために使用される。現在、これは、A_ENVELOPE、A_PANおよびA_RESAMPLEに先立って発生されるべきである。

【0188】[圧縮解除コマンド処理] 信号プロセッサ400により取り出された次のオーディオ再生リストコマンドが、圧縮解除コマンドの場合には、信号プロセッサは、圧縮解除操作を行い、データメモリ404内の入力バッファに記憶された圧縮オーディオバイナリストリームを圧縮解除し、16ビットのオーディオサンプルを生成し、データメモリ内の定義されたオーディオ出力バッファに16ビットオーディオサンプルを記憶する（図26、ブロック726）。この実施例では、オーディオ合成マイクロコード708は、以下のオーディオ圧縮解除コマンドフォーマットおよび関連機能をサポートする。

【0189】[圧縮解除コマンドの例]

A_ADPCM:

【表27】

コマンド	フラグ	ゲイン
セグメント		アドレス

このコマンドは、信号プロセッサデータメモリ404内のバイナリストリームを圧縮解除し、16ビットのサンプルを生成する。入力および出力バッファのためのデー

タメモリ404のアドレスおよび処理するサンプルの数は、A_SETBUFFコマンドを、A_ADPCMコマンドの前に発生することにより定義される。「セグメント」+「アドレス」フィールドは、状態を保存し、復元するために使用されるメインメモリ300の記憶場所を指定する。「ゲイン」パラメータは、出力を一定の基準とするために使用され、S. 15として表現される。

【0190】「フラグ」は、コマンドの機能を定義する。現在、定義されている「フラグ」は、以下の通りである。

A_INIT

コマンドの開始時に、状態を復元するために「セグメント」+「アドレス」フィールドが使用される。さもなくば、このフラグがセットされていないければ、状態を指すポインタは、起動時（開始時）には無視されるが、状態を処理の終わりにこのアドレスに保存する。

A_MIX

この結果は、出力バッファにミックスされる。さもなくば、このフラグがセットされていないければ、結果は、出力バッファに入力される。

【0191】[リサンプルコマンド処理] 信号プロセッサ400の次のオーディオ再生リストコマンドの読み出しが、リサンプルコマンドである場合には、信号プロセッサ400は、コマンドで特定されたパラメータに基づいて、統合包絡線変調に加え、ピッチシフティング/リサンプリングを提供する（図26、ブロック728）。以下は、オーディオ合成マイクロコード708によりサポートされた、リサンプルコマンドおよび関連機能の実施例である。

【0192】[リサンプルコマンドの例]

A_RESAMPLE:

【表28】

コマンド	フラグ	ピッチ
セグメント		アドレス

このコマンドは、統合包絡線変調に加えて、ピッチシフティング/リサンプリングを提供する。信号プロセッサのデータメモリ404の入力および出力バッファ、およびサンプルの数は、A_SETBUFFコマンドの発生により定義され、ボリューム包絡線パラメータは、A_SETVOLコマンドの発生により定義される。リサンプリング係数は、ピッチにより定義される。

【0193】「フラグ」は、コマンドの機能を定義する。現在、定義されているフラグは、以下の通りである。

A_INIT

コマンドの開始時に、状態を復元するために「セグメント」+「アドレス」フィールドが使用される。さもなくば、このフラグがセットされていないければ、状態を指すポインタは、起動時（開始時）には無視されるが、状

態を処理の終わりにこのアドレスに保存する。

A_MIX

この結果は、出力バッファにミックスされる。さもなくば、このフラグがセットされていないければ、結果は、出力バッファに入力される。

【0194】[包絡線/バンコマンド処理] 信号プロセッサ400の次のオーディオ再生リストコマンドの読み出しが、包絡線/バンコマンドである場合には、信号プロセッサ400は、線形包絡線を使用して、1つまたは2つのオーディオ信号ストリームを変調することにより、当該コマンドを実行する(図26、ブロック730)。包絡線コマンドは、線形関数によりオーディオ入力サンプルストリームを乗算し、これにより、オーディオの音量を上下にさせることができる。「バン」コマンドは、一般的に、左右のステレオチャンネルのオーディオに、逆線形関数を適用させ、それにより、空間において認識された音響または声の音源を移動する効果を達成する(すなわち、左から右または右から左への移動)。以下の包絡線/バンコマンドフォーマットおよび関連機能の実施例は、このシステム50の実施例では、オーディオ合成マイクロコード708によりサポートされている。

【0195】[包絡線/バンコマンドの例]

A_ENVELOPE:

【表29】

コマンド	フラグ	データメモリ出力 2
セグメント		アドレス

このコマンドは、線形包絡線を使用して、サンプルストリームを変調する。ボリューム包絡線のためのパラメータは、A_SETVOLを発生することにより定義され、信号プロセッサのデータメモリ404のバッファ記憶場所および処理するサンプルの数は、A_ENVELOPEコマンドを発生する前に、A_SETBUFFを発生することにより定義される。

【0196】「フラグ」は、コマンドの機能を定義する。現在、定義されているフラグは、以下の通りである。

A_INIT

コマンドの開始時に、状態を復元するために「セグメント」+「アドレス」フィールドが使用される。さもなくば、このフラグがセットされていないければ、状態を指すポインターは、起動時(開始時)には無視されるが、状態を処理の終わりにこのアドレスに保存する。

A_MIX

この結果は、出力バッファにミックスされる。さもなくば、このフラグがセットされていないければ、結果は、出力バッファに入力される。

【0197】A_PAN:

【表30】

コマンド	フラグ	データメモリ出力 2
セグメント		アドレス

このコマンドは、1入力/2出力バニングを提供する。入力、第1出力およびサンプルの数は、A_SETBUFFコマンドを発生することにより定義され、バニングパラメータは、A_SETVOLコマンドを発生することにより定義される。第2出力は、「データメモリ出力2」により定義される。

10 【0198】「フラグ」は、コマンドの機能を定義する。現在、定義されているフラグは、以下の通りである。

A_INIT

コマンドの開始時に、状態を復元するために「セグメント」+「アドレス」フィールドが使用される。さもなくば、このフラグがセットされていないければ、状態を指すポインターは、起動時(開始時)には無視されるが、状態を処理の終わりにこのアドレスに保存する。

A_MIX

20 この結果は、出力バッファにミックスされる。さもなくば、このフラグがセットされていないければ、結果は、出力バッファに入力される。

【0199】[ミキシングコマンド処理] 次のオーディオ再生リストのコマンドがミキシングコマンドである場合には、信号プロセッサ400は、2つのオーディオ入力サンプルストリームを出力オーディオバッファにミックスするミキシング機能を実行する(図26、ブロック732)。以下のミキシングコマンドのフォーマットおよび関連機能の実施例は、この実施例においては、信号プロセッサ400およびオーディオ合成マイクロコード708によりサポートされている。

【0200】[ミキサコマンドの例]

A_MIXER:

【表31】

コマンド	ゲイン
	データメモリ出力 f

このコマンドは、倍精度ミキシング機能を提供する。単精度入力は、「ゲイン」により乗算された後、倍精度出力に付加される。「データメモリ出力 f」は、ミックスされたストリームの小数部を記憶する信号プロセッサのデータメモリ404の領域を指定する。入力バッファ、サンプルの数およびミックスされた出力の整数部分は、A_MIXに先立ってA_SETBUFFを発生することにより定義される。

【0201】[特殊オーディオ効果処理] 次のオーディオ再生リストコマンドが特殊信号処理/効果コマンドである場合には、信号プロセッサ400は、指定された特殊効果または信号処理を提供することにより、コマンドを実行する(図26、ブロック734)。特殊信号処理

／効果の1つの実施例としては、臨場感を出すために反響を追加することである。この特殊効果は、洞窟、コンサートホール等における音の反響をシミュレーションしたものであり、他のさまざまな特殊効果にも使用することができる。信号プロセッサ400およびオーディオ合成マイクロコード708は、下記の反響特殊効果コマンドフォーマットおよび関連機能の実施例をサポートする。

【0202】[効果コマンドの例]

A_REVERB:

【表32】

セグメント	コマンド		フラグ	アドレス

このコマンドは、サンプルストリームに対して反響特殊効果を適用する。信号プロセッサのデータメモリ404の入力およびサンプルの数は、A_SETBUFFコマンドの発生により定義される。

【0203】「フラグ」は、コマンドの機能を定義する。現在、定義されているフラグは、以下の通りである。

A_INIT

コマンドの開始時に、状態を復元するために「セグメント」+「アドレス」フィールドが使用される。さもなくば、このフラグがセットされていないければ、状態を指すポインターは、起動時（開始時）には無視されるが、状態を処理の終わりにこのアドレスに保存する。

A_MIX

この結果は、出力バッファにミックスされる。さもなくば、このフラグがセットされていないければ、結果は、出力バッファに入力される。

【0204】[オーディオ処理の構造] この実施例では、各オーディオ処理機能728、730、732、734を達成するために、オーディオ合成マイクロコード708は、1つの遅延線でデータを処理する汎用効果実現を用いる。図27は、汎用オーディオ処理実現740の実施例を示している。この実施例において、オーディオ入力サンプルは、連続した1つの遅延線742の入力に適用されるものとして考えることができる。遅延線742の出力タップは、ゲイン744を介して、信号プロセッサのデータメモリ404内のオーディオ出力バッファに適用される。遅延線742上の別のタップからのサンプルは、加算器746を通して、遅延線742にパス748を介して直接戻り、また、係数ブロック750、別の加算器752およびローパスフィルタ754を介して遅延線742に戻る。遅延線742からさらに別のタップ756は、加算器752の他の入力に接続されており、また、加算器746他の入力に（この場合には、別の係数ブロック758を介して）接続されている。この汎用効果実現740は、1つの遅延線742に任意の数

のプリミティブを付加することにより、特定の効果が発生することを可能にする。効果中の各プリミティブに対するパラメータは、上述のコマンドを介して送られる。各プリミティブは、可変長タップを有する全てのバスから成り、DC正規化（DCで単一利得）単極ローパスフィルタ754がこれのあとに接続され、更に、このプリミティブの出力がどれくらいであるかを指定する出力ゲイン744がその後に接続され、これにより最終効果の出力が得られる。プリミティブに対するパラメータの各値は、当該効果内で全体としてのプリミティブの機能を特定する。図27を参照して、帰還係数758は、（上記で説明したa_reverbコマンドに応じて）「くし内全通過」反響を構成するのに使用することができる。

【0205】汎用効果実施740の一般的性質としては、すべての機能が、実行されるわけではない。正当なパラメータにより作動される機能のみが、信号プロセッサ400によるオーディオコマンド動作を生じさせる。このことは、ビデオゲームプログラムに、音質および音響効果の双方の面において、適切な効果を定義する際に、多大な融通性を与えている。

【0206】[コプロセッサ表示プロセッサ500] この実施例におけるデス再生プロセッサ500は、三角形および矩形をラスタライズし、テクスチャされ、アンチエイリアジングし、Zバッファリングされた高品質画素を生成する。図28は、表示プロセッサが実行した全体のプロセスを示している。表示プロセッサ500は、例えば提供される頂点、色、テクスチャ、表面法線および他のグラフィックスプリミティブの他の特徴を特定するグラフィックス表示コマンドを受け取る。この実施例において、表示プロセッサ500は、線、三角形および矩形を提供できる。一般的には、表示プロセッサ500は、信号プロセッサ400の提供するプリミティブの仕様を受け取る。また、メインプロセッサ100は、表示プロセッサに直接コマンドを指定することが可能である。

【0207】入来するプリミティブについての表示プロセッサ500の最初の動作は、プリミティブをラスタ化すること、すなわち、プリミティブ内部をカバーする画素を生成することである（図28、ブロック550）。ラスタライズしたブロック550は、多様な属性（例えば、画面位置、奥行、RGBA色情報、テクスチャ座標および他のパラメータ、並びに有効範囲値）をプリミティブ内の各画素のために生成する。ラスタライズしたブロック550は、テクスチャブロック552に、テクスチャ座標およびパラメータを出力する。テクスチャブロック552は、テクスチャメモリ502に記憶されているテクスチャ情報をアクセスし、テクスチャメモリ内の特定のテクスチャのテクセル（テクスチャ要素）を、ラスタライズしたブロック550が出力した各画素に適用（「マップ」）する。色変換ブロッ

ク554およびクロマキーイングブロック556は、更に、テクスチャカラーをカラーコンバインブロック558に提供するための画素値を処理する。

【0208】ところで、ラスタライズしたブロック550は、同じ画素のための、プリミティブカラー（例えば、シェーディングの結果として）をカラーコンバインブロック558に提供する。カラーコンバインブロック558は、これら2つの色を組み合わせ、1つの信号画素色にする。この1つの画素色出力は、ブロック560によって、画素色出力に適用されるフォグを有してもよい（例えば、煙が充満した部屋、または、視覚者から物体が離れていくにしたがって、色の輝きを減少する極端すぎない自然な効果を醸し出す目的で）。得られた画素色値は、ブロック562によりブレンドされ、一方、画素値フレームバッファ118は、同一スクリーン上の座標位置を保存する。付加のアンチエイリアシング／Zバッファ操作564は、隠面除去（すなわち、近くにある不透明物体が遠くにある物体を隠すこと）およびアンチエイリアシング（一連の画素により近似されたプリミティブのエッジ部分のギザギザを除去すること）を実行し、新しい画素値をフレームバッファ118へ再び書き込ませる。

【0209】図28に示す操作は、各プリミティブ内の各画素を提供するために行われる。多数のプリミティブは、1つの複合シーンを定義し、また各プリミティブは、何百または何千の画素を含むことができる。このように、表示プロセッサ500は、カラーテレビ受像機58に表示される各イメージ毎に何百万もの画素を処理しなければならない。

【0210】一般的には、フレームバッファ118は、「ダブルバッファされている」、つまり、2つの完全なテレビ画面の画像を含むように規格化されている。表示プロセッサ500は、1画面相当のフレームバッファ情報を書き込み、一方、ビデオインタフェイス210は、フレームバッファ118の他の半分を読み取る。ビデオフレームの終わりに、ビデオインタフェイス210は、表示プロセッサ500と場所を交換して、ビデオインタフェイス210は、表示プロセッサ500が完了したばかりの新しい画像表示を読み取り、表示プロセッサがフレームバッファの他の半分の再び書き込む。このダブルバッファリングは、表示プロセッサ500に画像を完成する時間をこれ以上与えない。さらに、表示プロセッサ500は、通常1ビデオフレーム時間内（すなわち、新たな画像が表示されるフレーム時間の直前のビデオフレーム時間中）に画像を終了しなければならない。

【0211】「バイブライン処理」画素を提供するためには、高速度操作が非常に重要であるため、表示プロセッサ500は、「バイブライン」として動作するように設計されてきた。図28に再び戻ると、「バイブライン」とは、図28に示す多数のステップが、異なった

画素について平行して実行できることを意味する。例えば、ラスタライズブロック550は、テクスチャブロック552に最初の画素値を与え、そしてテクスチャブロックが最初の画素値についてまだ作業している間に、次の画素値についての作業を開始することができる。同様に、ラスタライズブロック550は、ブレンドブロック562が作業している画素よりも、多くの画素分だけ先行していてもよい。

【0212】この実施例では、表示プロセッサ500は、1サイクルモードおよび2サイクルモードの2つの異なったバイブラインモードを有している。1サイクルモードにおいては、1つの画素が、表示プロセッサ500の各サイクルタイム期間ごとに処理される。1サイクルモード操作は、図29(a)に示されている。図29(a)に示された操作自身がバイブライン処理されるが（すなわち、ブレンド操作562は、ラスタライズ操作550が現在ラスタライズしている画素以外の画素を操作する）、全体の操作シーケンスは、サイクル当たり1画素を処理する。

【0213】図29(b)は、この実施例における表示プロセッサ500の2サイクルバイブラインモードの操作を示している。図29(b)の実施例には、図28に示された操作の幾つかが、各画素につき2回実行されている。例えば、図28に示されたテクスチャおよび色変換／フィルタリング操作552、554は、各画素毎に繰り返され、カラーコンバイン操作558は、2回実行されている（あるテクスチャ操作のテクスチャカラー出力につき1回、また他のテクスチャ操作のテクスチャカラー出力につき1回）。同様に、図28に示すブレンド操作562は、各画素毎に2回行われている。

【0214】これらの多様な操作が2回行われたとしても、本実施例における表示プロセッサ500は、コピー操作を同時に行うコピーハードウェアを含まない（かかるハードウェアの複製は、費用と複雑性を増加させるためである）。従って、この実施例においては、表示プロセッサ500は、特定の回路（例えば、テクスチャユニット、カラーコンバインあるいはブレンド）を用いて1つの画素を処理することによって、画素に対する操作を複製し、同一の画素について同じタイプの操作を実行するために、同一の回路を再び使用する。この繰り返しは、2つの要因（各画素は、1つのサイクルではなく、2つのサイクル毎にバイブラインにおける各ストップに「とどまらなければならない」によりバイブライン処理を低減させるが、より複雑な処理ができる。例えば、画素当たり2サイクルのモードは、同一の画素に2つのテクスチャをマップできるため、「トリリニア」

（「ミップマッピング」）テクスチャマッピングをなすことが可能となる。更に、この実施例では、表示プロセッサ500は、フォグ操作560およびブレンド操作

562の双方を行うために、同一のブレンダハードウェアを使用している（しかし、ブレンドおよびフォグを同時にできない）ので、通常、有効なフォグ効果を提供するために、画素当たり2サイクルのモードを操作する必要がある。

*【0215】以下の表は、1サイクルおよび2サイクルモード中の図29(a)および図29(b)に示された、多数のブロックによって行われる操作を要約している。

*【表33】

1サイクルモードにおけるディスプレイプロセッサのパイプラインブロック機能	
ブロック	機能
ラスタライズ 550	画素及びそのプリミティブ内部によりカバーされた属性を生成する。
テクスチャ 552	テクスチャマップ内の画素に最も近い4つのテクセルを生成する。
フィルタ テクスチャ 554	4つのテクセルを1つのテクセルにバイリニアフィルタリングする、又は、YUV-RGB変換のステップ1を実行する。
コンバイン 558	多数の色を1つの色に結合させる、又は、YUV-RGB変換のステップ2を実行する。
ブレンド 562	画素をフレームバッファメモリ画素でブレンドする、又は、フレームバッファに書き込むために画素をフォグする。
フレームバッファ 563	フレームバッファメモリから画素（色及びz）を取り出し、かつ、これをフレームバッファメモリに書き込む。

【表34】

2サイクルモードにおけるディスプレイプロセッサのパイプラインブロック機能	
ブロック	機能
ラスタライズ 550	画素及びそのプリミティブ内部によりカバーされた属性を生成する。
テクスチャ 552a	テクスチャマップ内の画素に最も近い4つのテクセルを生成する。これはミップマップのレベル X にできる。
テクスチャ 552b	テクスチャマップ内の画素の最も近い4つのテクセルを生成する。これはミップマップのレベル X+1 にできる。
フィルタ テクスチャ 554a	バイリニア、4つのテクセルを1つのテクセルにフィルタする。
フィルタ テクスチャ 554b	バイリニア、4つのテクセルを1つのテクセルにフィルタする。
コンバイン 558a	多数の色を1つの色に結合させるか、ミップマップの2つの隣接されたレベルからの2つのバイリニアフィルタリングされたテクセルを線形補間するか、又は、YUV-RGB変換のステップ2を実行する。
コンバイン 558b	多数の色を1つの色に結合されるか、又は、クロマキーイングする。
ブレンド 562a	フォグカラー合成 OC1 カラーと結合させる。
ブレンド 562b	パイプラインの画素をフレームバッファメモリの画素とブレンドする。
フレームバッファ 563a	色メモリを読み出し/修正/書き込みする。
フレームバッファ 563b	zメモリを読み出し/修正/書き込みする。

【0216】【フィルおよびコピー操作】表示プロセッサ500は、更に「フィル」モードおよび「コピー」モ

ードを有し、その各々は、1サイクル毎に4つの画素を処理する。フィルモードは、フレームバッファ118のエリアを、同一の画素値で満たすのに使用される(例えば、フレームバッファまたはそのエリアを高性能クリアするために)。コピーモードは、高性能画像-画像コピーに(例えば、表示プロセッサテクスチャメモリ502から、フレームバッファ118の特定エリアへのコピーに)使用される。コピーモードは、他の方向へ(すなわち、フレームバッファからテクスチャメモリへ)の高性能コピーの提供に加えて、ビット「ブリット」を提供する。

【0217】図29(a)および図29(b)に示すパイプライン操作は、この実施例においては、画素フィルまたはコピーレートに操作が追従できないので、フィルおよびコピーモード中にはほとんど不使用である。しかし、この実施例において、「アルファ比較」操作(ブレンド操作562の一部)は、コピーモードで動作可となり、表示プロセッサ500が、画像をフレームバッファ118に「ブリット」し、また、条件付きで画像画素をワードアルファ=0(例えば、透明画素)により除去することが可能である。

【0218】表示プロセッサの操作モードは、表示プロセッサ500に、「サイクルタイプ」パラメータを指定する「他のモードのセット」コマンドを送ることにより選択される。図86参照。画素当たり1サイクルまたは画素当たり2サイクルのパイプラインモードにおいて、追加の表示プロセッサ500コマンドは、パイプライン同期が維持されることを保証するのに有効である(すなわち、それにより別のプリミティブのパラメータが効果を生ずる前に、パイプラインから1つのプリミティブを取り除く)。図100を参照。

【0219】[表示プロセッサ500のアーキテクチャの実施例]図30は、表示プロセッサ500のアーキテクチャの実施例を示している。この実施例においては、表示プロセッサ500は、RAM516およびDMAコントローラ518に関するコマンドユニット514、「エッジウォーク」/ラスターライザ504、RGB AZ画素ステッパ520、カラーコンバイナ/レベルインタブリタ508、ブレンダ/フォガー510、ディザラ522、有効範囲エバリュータ524、奥行(Z)比較器526、メモリアインタフェイス512、およびテクスチャユニット506を含んでいる。この実施例では、テクスチャユニット506は、テクスチャメモリ502に加えて、テクスチャステッパ528、テクスチャ座標ユニット503、およびテクスチャフィルタユニット532を含んでいる。

【0220】コマンドユニット514およびDMAコントローラ518は、コプロセッサメイン内部バス214に接続し、更に、プライベート「X」バス218を介して、信号プロセッサ400に接続している。メモリア

タフェイス512は、特別なメモリアインタフェイスであり、メインメモリ300内に記憶されたカラーフレームバッファ118aおよびZバッファ118bに主にアクセスするために、表示プロセッサ500によって使用される(従って、表示プロセッサ500は、メモリアインタフェイス512を介して、更に、コプロセッサ内部バス214を介して、メインメモリ300にアクセスする)。

【0221】[DMAコントローラ]DMAコントローラ518は、バス214を介して信号プロセッサ400またはメインプロセッサ100からDMAコマンドを受け取る。DMAコントローラ518は、図31(a)~図31(c)に示す読み出し/書き込みレジスタを有しており、信号プロセッサ400および/またはメインプロセッサ100に、SPデータメモリ404またはメインメモリ300において、そこからグラフィックス表示コマンドの一連を読み出すために、スタートおよびエンドアドレスを指定させる(図31(a)は、スタートアドレスレジスタ518Aを示し、図31(b)は、エンドアドレスレジスタ518Bを示している)。DMAコントローラ518は、レジスタ518a、518bがメインメモリ300のアドレスを特定する場合には、メインコプロセッサバス214を介してデータを読み出し、そして、レジスタ518a、518bがデータメモリ404のアドレスを特定する場合には、データをプライベート「Xバス」214を介して信号プロセッサのデータメモリ404からデータを読み出す。DMAコントローラ518もまた、DMAコントローラ518が読み出す現在のアドレスを含むさらなるレジスタを含んでいる

(図31(c)に示されたレジスタ518C)。この実施例において、DMAコントローラ518は、単一方向性であり、すなわち、バス214からRAM516にのみ書き込みができる。このように、DMAコントローラ518は、本実施例においては、信号プロセッサ400またはメインメモリ300からの読み出しのために使用される。この実施例において、表示プロセッサ500は、テクスチャロードコマンドをコマンドユニット514に渡し、メモリアインタフェイス512を使用して、これらのコマンドを実行させることによって、テクスチャメモリ502のためのデータを取得する。

【0222】[コマンドユニット]コマンドユニット514は、表示プロセッサ500に関する現行状態情報(例えば、「コマンドのセット」により特定されたモードおよび他の選択)の大部分を保有し、表示プロセッサ500の残りの操作を特定し、決定する属性およびコマンド制御信号を出力する。コマンドユニット514は、コプロセッサバス214経由でメインプロセッサ100(または信号プロセッサ400)によりアクセスできる幾つかの追加レジスタを含む。メインプロセッサ100のアドレススペースにマップされたこれらの追加レジス

タは、メインプロセッサが表示プロセッサ500を制御し、モニターすることを可能にしている。

【0223】例えば、コマンドユニット514は、図32.(d)に示されたステータス/コマンドレジスタ534を含んでおり、これはメインプロセッサ100に読み出されるときは、ステータスレジスタとして機能し、メインプロセッサがそれに書き込みをするときは、コマンドレジスタとして機能する。このレジスタ534を読み出すときは、メインプロセッサ100は、以下を決定することができる。つまり、表示プロセッサ500は、信号プロセッサのデータメモリ404（フィールド536（1））から読み出すDMA操作を行っていて使用中であるか、表示プロセッサがメインメモリ300にアクセス待ちのため停止させられているか（フィールド536（2））、表示プロセッサのパイプラインはフラッシュされたか（フィールド536（3））、表示プロセッサのグラフィッククロックは開始させられたか（フィールド536（4））、テクスチャメモリ502はビジーか（フィールド536（5））、表示プロセッサのパイプラインはビジーか（フィールド536（6））、コマンドユニット514はビジーか（フィールド536（7））、コマンドバッファRAM516は新たな入力を受け入れる用意があるか（フィールド536（8））、DMAコントローラ518はビジーか（フィールド536（9））、スタートおよびエンドアドレス並びにレジスタ518aおよび518bは各々有効か（フィールド536（10）、536（11））を決定できる。この同一のレジスタ534に書き込むときに、メインプロセッサ100（または信号プロセッサ400）は、信号プロセッサ400からのXバスDMA操作をクリアしたり（フィールド538（1））、信号プロセッサ400からのXバスDMA操作を開始したり（フィールド538（2））、表示処理を開始または停止したり（フィールド538（3））、フィールド538（4））、パイプラインフラッシング操作を開始または停止したり（フィールド538（5））、フィールド538（6））、図33（h）に示されたテクスチャメモリアドレスカウンタ540をクリアしたり（フィールド538（7））、図33（f）に示されたパイプラインビジーカウンタ542をクリアしたり（フィールド538（8））、コマンドバッファRAM516を指すために使用されるコマンドカウンタ544をクリアしたり（フィールド538（9））（カウンタ544は、図33（g）に示されている）、クロックサイクルをカウントするために使用されるクロックカウンタ546（図33（e）参照）をクリアする（フィールド538（10））ことができる。

【0224】上記の通り、クロックカウント、バッファカウント、パイプラインカウント、およびテクスチャメモリカウントは、すべてレジスタ540～546から

直接読み出すことができる（図33（e）～図33

（h）参照）。更に、メインプロセッサ100または信号プロセッサ400は、テクスチャメモリ502に関するBIST操作を読み出して制御することができ（図34（i）に示されたBISTステータス/制御レジスタを参照）、図34（j）に示されたメモリスバンテストレジスタ549（a）、549（b）および549（c）を操作することにより、メモリインタフェース512のテストを制御することを可能にすることができる。

【0225】図30に戻り、1つまたはそれ以上のコマンドは、コマンドユニットバッファRAM518にロードされ、表示プロセッサ500が開始され、コマンドユニット514は、各コマンドの読み出しおよび処理を順次始める。表示プロセッサ500が理解するコマンドのレパートリは、図65～図102に記載している。表示プロセッサ500内のハードウェア（例えば、ロジック、ゲートアレイ等）は、RAM516内のグラフィックス表示コマンドを直接解釈する。この実施例では、表示プロセッサ500は、このコマンドのリストを飛ばして、分岐またはジャンプする能力はない。むしろ、この実施例においては、表示プロセッサ500は、各新たなコマンドを厳密なシーケンス中の入力として受けとり、そしてコマンドに応じて、その状態を変更し出力するシーケンス状態の装置である。

【0226】表示プロセッサ500は、そのコマンドバッファRAM516が空の場合には停止する（すなわち、バッファ内の全てのコマンドを処理したのであり、そのバッファは、FIFOとして行為をなす）。メインプロセッサ100または信号プロセッサ400は、表示プロセッサ500が、表示プロセッサのステータスレジスタ534を読み出すことにより停止したかどうかを決定することができ、所望する場合には、表示プロセッサを一時的に停止させるコマンドを表示プロセッサに伝えることができる（図98参照）。

【0227】[エッジウォークおよびステップ] 図30に示されたエッジウォーク504は、図28に示されたラスタライズ処理550を行う。この実施例においては、エッジウォーク504は、「三角コマンド」で特定されたエッジ係数、シェード係数、テクスチャ係数、およびZバッファ係数を受け取り（特定のプリミティブオープンライン、三角形または矩形を指定する図73を参照）、「スパン」値を出力し、当該「スパン」値からプリミティブに内蔵された各画素ごとに、下記の属性が引き出される。

- ・スクリーンx、yの位置
- ・Zバッファ目的のためのZ奥行
- ・RGBA色情報
- ・s/w、t、w、1/wテクスチャ座標、テクスチャインデックスの詳細レベル、遠近修正およびミップ

マッピングに用いられる通常 s 、 t 、 w 、 1 、と称される)・有効範囲値(プリミティブ内部の画素は全体的であるが、プリミティブの端にある画素は、部分的な有効範囲値を有する)

【0228】エッジウォーク504は、プリミティブを横切る画素のラインのパラメーター(「スパン」)を、他の計算のためにパイプラインハードウェアダウンストリームに送る。特に、テクスチャステッパ528およびRGBAZステッパ520は、エッジウォーク504に特定された「スパン」情報を受けとり、(ビュー平面座標システムにおいて)「スパン」の水平線上の各画素に順次沿って進み、スパンにおける各個別の画素ごとに個別のテクスチャ座標およびRGBAZ値を引き出す。

【0229】RGBAZステッパ520は、さらに、三角形プリミティブに「シザリング」操作を実行し(本実施例では、この操作は矩形に対しては作用しない)、ビュー平面シザリング矩形の外側に延びる三角プリミティブの部分を、効果的に除去する。シザリングは、普通は、信号プロセッサ400において、ランニングパフォーマンス-集中クリッピング操作を除去するために使用される。シザリングは、概念においては、クリッピングと類似しているが、クリッピングは3D座標システムにおいて実行されるが、シザリングは、ビューイング平面の2D座標システムにおいて実行される。ステッパ520、528によるシザリングは、表示プロセッサ500に「シザーのセット」コマンドを送ることにより呼び出される(図94参照)。

【0230】上記のように、ステッパ520は、エッジウォーク504により定義される「スパン」内の各画素ごとに、色およびアルファ情報を生成する。同様に、テクスチャステッパ528は、スパン内の各画素のテクスチャ座標値(s 、 t 、 w)を生成する。ステッパ520、528は、RGBAZステッパ520がプリミティブカラー、シェーディング、ライティング等に基づいて、画素の色値を出力すると同時に、テクスチャユニット506が、マッピングされたテクスチャ値をカラーコンバイナ58に出力するよう同期して動作する。

【0231】[テクスチャユニット]この実施例におけるテクスチャユニット506は、画素のテクスチャ座標 s 、 t 、 w および詳細レベル値を取り(上述のとおり、テクスチャステッパ528は、エッジウォーク504が提供した「スパン」情報に基づいて、各個別の画素ごとにこれらの値を引き出す)、画素にマッピングするために実装のテクスチャメモリから、適切なテクスチャ情報を取り出すものである。この実施例においては、画面画素に最も近い4つのテクセルは、テクスチャメモリ502から取り出され、これらの4つのテクセル値は、マッピング目的のために使用される。ビデオゲームプログラム108は、テクスチャ画像タイプお

よびフォーマット等のテクスチャ状態と、テクスチャ画像をどのようにどこでロードするかということと、テクスチャサンプリング属性とを操作できる。

【0232】テクスチャ座標ユニット530は、テクスチャメモリ502内に記憶されたテクスチャをマッピングするために提供されたプリミティブ上で、適切なテクスチャ座標を計算する。テクスチャメモリ502に記憶された2次元テクスチャは、多様なサイズの三角形上にマップされなければならない四角形または矩形画像であるので、テクスチャ座標ユニット530のテクスチャ座標は、プリミティブ内の画素にマッピングするテクスチャ内の適切なテクセルを選択し、テクスチャを歪めるのを避けるなければならない。オープンGLプログラミング・ガイドの278頁を参照。

【0233】テクスチャ座標ユニット530は、入力された画素テクスチャ座標とテクスチャメモリ502に記憶された適切なテクスチャの内の4つのテクセルの間のマッピングを計算する。テクスチャ座標ユニット530は、それから、これら4つのテクセルを適切に検索するために、テクスチャメモリ502をアドレス指定する。4つのテクセル値は、テクスチャフィルタユニット532に渡される。テクスチャフィルタユニット532は、テクスチャメモリ502から検索した4つのテクセルを取り、簡単なバイリニアフィルタされたテクセルを生成する。この実施例のテクスチャフィルタ532は、ポイントサンプリング、ボックスフィルタリングおよびバイリニア補間の3種類のフィルタ操作を実行できる。ポイントサンプリングは、画面画素に最も近いテクセルを選択する。画面画素が常に4つのテクセルの中央にあるという特別な場合には、ボックスフィルタを使用することができる。典型的な3D、任意に回転する多角形の場合には、バイリニアフィルタリングは、一般的に利用可能な最善の選択である。ハードウェア費用を軽減するために、表示プロセッサのテクスチャフィルタユニット532は、真正なバイリニアフィルタを実施しない。代わりに、それは3つの最も近接したテクセルを直線的に保管し、画素を生成する。これは通常のテクスチャ画像では目立たないが、規則的なパターン画像では顕著な自然三角バイアスを有する。このアーチファクトは、幅広いフィルタによりテクスチャ画像をブリフィルタリングすることにより除去できる。テクスチャフィルタユニット532により行われるフィルタリングのタイプは、「モードのセット」表示コマンド内のパラメータを使用して設定される(図65~図102を参照)。

【0234】[テクスチャメモリ502]表示プロセッサ500は、テクスチャメモリ502を汎用テクスチャメモリとして扱っている。この実施例においては、テクスチャメモリ502は、1クロックサイクル当たり4テクセルの出力を与える4つの同時にアクセス

可能なバンクに分割されている。ビデオゲームプログラム58は、テクスチャメモリ502の至る所で、異なったフォーマットのサイズ可変のテクスチャをロードできる。テクスチャ座標ユニット503は、テクスチャメモリ502におけるテクスチャ画像の位置と、各テクスチャのフォーマットと、およびそのサンプリングパラメータとを記載する8つのテクスチャタイル記述子を保持している。このことにより、表示プロセッサ500は、1回に8つもの異なったテクスチャタイルにアクセスすることができる(8つ以上のテクスチャタイルを、テクスチャメモリにロードすることができるが、何時でもアクセス可能なのは、8つのタイルだけである)。

【0235】図35は、テクスチャタイル記述子およびテクスチャメモリ502に記憶されているテクスチャタイルとの関係の実施例を示している。図35に示されたこの特定の実施例では、8つの異なったテクスチャタイル802は、テクスチャメモリ502に保存されている。テクスチャタイル802は、それぞれ関連するテクスチャタイル記述子ブロック804を有している(上記で述べたように、表示プロセッサ500は、テクスチャメモリ502に記憶されている8つのテクスチャタイルに対応する記述子804を、8つまで保持している)。テクスチャ記述子は、「タイルのセット」コマンドにより特定された情報を含んでいる(図68参照)。例えば、これらのテクスチャタイル記述子は、画像データフォーマット(RGBA、YUV、カラーインデックスモード等)、各画素/テクセルの色要素のサイズ(4、8、16、32ビット)、64ビットワードのタイルラインのサイズ、テクスチャメモリ502内のタイルの開始アドレス、4ビットカラーインデックス化されたテクセルのパレット番号、SおよびT方向のそれぞれに可能なクランプおよびミラー、SおよびT方向のそれぞれにおけるラッピング/ミラーリングのマスク、SおよびTアドレスのそれぞれに対する詳細レベルのシフトを特定している。これらの記述子804は、テクスチャメモリ502内のテクセルのアドレスを計算するために、テクスチャ座標ユニット530により使用される。

【0236】[テクスチャ座標ユニット] 図36は、テクスチャ座標ユニット530が行う処理のより詳細な実施例を示している。図36は、テクスチャ座標ユニット530への入力として適用される多様なタイル記述子804を示している。図36は、さらに、テクスチャ座標ユニット530が、テクスチャステップ528から、現時点の画素のためのプリミティブタイル/レベル/テクスチャ座標を受け取る。テクスチャ座標ユニット530は、例えば「他のモードのセット」および「テクスチャ画像のセット」コマンドに基づいて、コマンドユニット514からモード制御信号を受け取る

(図86、図87および図66参照)。この入力情報のすべてに基づいて、テクスチャ座標ユニット530は、このプリミティブに使用するタイル記述子804を計算し、入力されたテクスチャ画像座標をタイル相対座標に変換する。当該タイル相対座標は、タイル記述子804により特定されるように、テクスチャ座標ユニットによってラップ、ミラーおよび/またはクランプされる。テクスチャ座標ユニット530は、それからこれらのタイル座標に基づき、テクスチャメモリ502にオフセットを生成する。この実施例のテクスチャ座標ユニット530は、1または2サイクルモードでは2×2領域を、若しくはコピーモードでは4×1領域をアドレス指定できる。テクスチャ座標ユニット530は、さらに、テクセルをバイーリニアリまたはトライーリニアリに補間するために使用するS/T/L端数値を生成する。

【0237】図37は、テクスチャ座標ユニット530およびテクスチャメモリユニット502の詳細図である。図37に示されるように、入来するs、t、wテクスチャ座標は、遠近修正が可能な時に、wに基づく遠近修正を提供する遠近修正ブロック566に入力される。遠近修正されたs、t値は、遠近分割の後に(例えば、ミップマッピングのために、および可能ならば精密にするために)テクスチャ座標を移動させる詳細レベルの、または精密なシフトブロック568に提供される。ブロック570は、その後、移動されたテクスチャ座標をタイル座標に変換し、端数値をテクスチャフィルタユニット532に提供する。これらのタイル座標は、表示プロセッサ500の現時点のテクスチャモードパラメータに基づいて、ブロック572によって、クランプ、ラップおよび/またはミラーされる。それとは別に、遠近修正ブロック566により提供された遠近修正テクスチャ座標は、また、詳細レベルブロック574に提供され、詳細レベルの計算が可能となったときは、詳細レベルブロック574は、タイル記述子インデックスをタイル記述子メモリ576に算入し、さらにカラーコンバイナ508による補間のために、詳細レベルの小数値を計算する。タイル記述子804は、タイル記述子メモリ576に記憶され、検索され、メモリ変換ブロックに出力され、当該メモリ変換ブロック578は、また、ブロック572の調整されたテクスチャ座標値を受け取る。アドレス変換ブロック578は、調整されたテクスチャ座標値を、タイル記述子804に特定された、現行タイルサイズ、フォーマットおよび他のパラメータに基づいて、テクスチャメモリユニットアドレスに変換する。アドレス変換ブロック578は、テクセルアドレスをテクスチャメモリユニット502に出力する。テクスチャメモリユニット502は、また例えば、テクスチャがカラーインデックスされた場合に使用される追加パラメータを受け取る。テクスチャメ

メモリユニット502は、上記で述べたように、4つのテクセル値をフィルタリングするために、テクスチャフィルタユニット532に出力する。

【0238】[テクスチャメモリローディング] テクスチャメモリユニット502は、コプロセッサ200に実装の4キロバイトのランダムアクセスメモリを含むものである。テクスチャは、一貫したアクセス時間で大量のランダムアクセスを要求するために、この実施例において、メインメモリ300から直接テクスチャするのは実行不可能である。方法としては、ワンチップ高速テクスチャメモリ502において、4キロバイトまで画像をキャッシュすることである。すべてのプリミティブは、テクスチャメモリ502の内容を使用してテクスチャできる。

【0239】テクスチャメモリ502を使用するために、ビデオゲームプログラム108は、テクスチャタイルをテクスチャメモリにロードし、関連する記述子804をタイル記述子576にロードしなければならない。「タイルのロード」コマンド(図69参照)は、タイルをテクスチャメモリ502にロードするのに使用され、「タイルのセット」および「タイルサイズのセット」コマンドは、対応するタイル記述子ブロック804をタイル記述子メモリ576にロードするのに使用される。更に、「タルトのロード」コマンド(図72参照)は、テクスチャルックアップテーブルを、カラーインデックステクスチャによって使用するために、テクスチャメモリ502にロードするために使用できる。

【0240】物理的には、テクスチャメモリ502は、4つのバンクに組織され、各バンクは、256の16ビット幅のワードからなり、ローハーフおよびハイハーフを有している。この組織は、4ビットテクスチャ(1列当たり20テクセル)、8ビットテクスチャ(1列当たり10テクセル)、16ビットテクスチャ(1列当たり6テクセル)、16ビットYUVテクスチャ(1列当たり12テクセル)および32ビットテクスチャ(1列当たり6テクセル)を記憶するのに使用できる。更に、この実施例におけるテクスチャユニット506は、テクスチャメモリ502のハイハーフが、カラールックアップテーブルを記憶するために使用され、テクスチャメモリ520のローハーフが、4ビットまたは8ビットカラーインデックステクスチャを記憶するために使用されるカラーインデックステクスチャモードをサポートする。この組織は、図38に示されている。この図38の実施例では、カラーインデックステクスチャタイル580は、テクスチャメモリ5

02のローハーフ502(L)に記憶され、対応するカラールックアップテーブル582は、テクスチャメモリのハイハーフ502(H)に記憶される。

【0241】図39は、特定のテクスチャメモリカラーインデックスモードをより詳細に示しており、当該モードにおいて、カラールックアップテーブル582は、例えば、各々が16ビット幅である16のエントリを有する4つのバレットバンク584またはテーブルに分割される。カラールックアップテーブルは、16ビットRGB Aフォーマットまたは16ビットIAフォーマットで色を表現する。4つのテクセルは、同時にアドレスされるので、テクスチャメモリ502のハイハーフに記憶される4つの(通常は同一)ルックアップテーブル484が存在する。上記のように、これらのルックアップテーブルは、図72に示す「タルトのロード」コマンドを使用してロードされる。

【0242】表示プロセッサ500は、テクスチャメモリ502のローハーフにある各テクセルが、8ビットを構成する別のカラーインデックステクスチャモードをサポートしており、そのため、テクスチャメモリ502のハイハーフ502(H)の256位置のいずれか1つに直接アクセスできる。よって、8ビットカラーインデックスされたテクスチャは、全体の256要素ルックアップテーブルに直接アドレスするので、タイルのバレット番号を使用しない。8ビットカラーインデックステクスチャを使用するとき、ルックアップテーブル用にテクスチャメモリ502のハイハーフを使用する必要はない。例えば、カラールックアップに8ビットカラーインデックステクスチャタイルの8個未満のビットが使用された場合には、カラーメモリのハイハーフ502(H)の一部だけがルックアップを保存することを要求され、テクスチャメモリ502の残りのハイハーフは、4ビットIテクスチャなどのノンカラーインデックステクスチャの保存のために使用されてもよい(図38参照)。同様に、カラーインデックステクスチャ580がテクスチャメモリ502のローハーフ502(L)に保存されたときも、それと同じくノンカラーインデックステクスチャをローハーフにも保存することも可能である。従って、カラーインデックステクスチャおよびノンカラーインデックステクスチャは、テクスチャメモリ502に共存できる。

【0243】下記のテクスチャフォーマットおよびサイズは、テクスチャメモリ502およびテクスチャ座標ユニット530によりサポートされている。

【表35】

テクスチャフォーマットおよびサイズ				
タイプ	4ビット	8ビット	16ビット	32ビット
RGBA			✓	✓
YUV			✓	
カラーインデックス	✓	✓		
強度アルファ(IA)	✓	✓	✓	
強度(I)	✓	✓		

【0244】この実施例において、テクスチャユニット506は、たとえ前のプリミティブのテクスチャマッピングにまだ使用されていても、明白に別段の記載ない限り、ローディング直後にタイル記述子804またはテクスチャタイル802を変更する。プリミティブレンダリング後のテクスチャロードは、「ロードのシンク」コマンドの後で実行され、タイル記述子の属性変更は、「タイルのシンク」コマンドの後で実行されるべきであり、それにより、テクスチャユニット506のテクスチャタイルおよびタイル記述子状態が、最後のプリミティブの処理が完全に終了される前に変更しないようにする（これらのコマンドのフォーマットおよび機能の実施例として、図99、図101を参照）。

【0245】信号プロセッサ400に関連して、上述したように、2つの特別なコマンド（「矩形のテクスチャ」および「矩形フリップのテクスチャ」）は、矩形プリミティブにテクスチャをマップするために使用できる（図83、図84参照）。「矩形のテクスチャ」コマンドは画像を、例えば、テクスチャメモリ502から、フレームバッファ118にコピーするために使用することが可能である。図83参照。

【0246】「カラーコンバイナ」図30を再び参照して、カラーコンバイナ508は、テクスチャユニット506により出力されたテクセルを、RGBAZステップ520により出力されたステップ処理済みRGBA画素値と組み合わせる。カラーコンバイナ508は、多数のソースから2つの色値を取り、その間で線形に補間できる。カラーコンバイナ508は、以下の式を実行する。

$$\text{新しい色} = (A - B) \times C + D$$

ここでは、A、B、CおよびDは、多くの異なったソースから得ることができる（D=Bの場合には、カラーコンバイナ508は、簡単な線形補間を実行する）。

【0247】図40は、この実施例では、RGBとアルファカラーとを組み合わせるための汎用線形補間カラーコンバイナ508の可能な入力選択を示している。図40でみられるように、左側の欄の入力のうちの一部だけが、テクスチャユニット506またはRGBAZステップ520から来ている。残りの入力、コマンドを表示プロセッサ500に送ることによりプログラムできるカラーコンバイナ508の内部状態から引き出したもの

である。上述したように、カラーコンバイナ508に提供される「コンバインカラー」および「コンバインアルファ」値は、RGBAZステップ520から得られ、テクセルカラーおよびテクスチャアルファは、テクスチャユニット506から得られる。（1画素当たり2サイクルモード2テクセルでは、例えば、ミップマッピングの目的のためにテクスチャユニット506によって提供されるので、2つのテクセルカラーおよび相応するアルファ値が示されている）。更に、詳細な端数入力のレベルは、図37のブロック574から得られ、プリミティブカラーおよびプリミティブアルファ値に加えて、詳細値のプリミティブレベルは、表示プロセッサ500に送られた「プリミティブカラーのセット」コマンド（図89参照）から得られる。（詳細端数値のプリミティブカラー値/アルファ/レベルは、不変の多角形表面色を設定するのに使用できる）。同様にシェードカラーおよび関連アルファ値は、「シェードの係数」コマンド（図77、図78参照）から得ることができ、環境カラーおよび関連アルファ値は、「環境カラーのセット」コマンド（図88参照）から得ることができる（上記に記載された環境カラー/アルファ値は、環境の周囲の色を表現するために使用できる）。「キーのセット」コマンドの2つの種類（1つは緑/青用、他の一方は赤用）は、各々緑/青色キーイングおよび赤色キーイングに使用されている。これらは、適切なキーセンターおよびキースケール入力をカラーコンバイナ508に提供している（図96、図97参照）。プリミティブおよび環境値の双方は、プログラム可能であり、従って、一般線形補間ソースとして使用できる。

【0248】カラーコンバイナ508への変換K4およびK5入力は、この実施例においては、YUVからRGBフォーマットへのテクセル値の変換の赤色座標を調整する「コンバートのセット」コマンド（図95参照）によって特定されている（このコンバートのセットに応答する残りの変換処理は、テクスチャフィルタユニット532内で行われる）。

【0249】図41は、図40において入力として示されるアルファ値を組み合わせるために使用されるカラーコンバイナ508の一部を示している。カラーコンバイナ508により実行されるアルファカラーコンバイン操作におけるRGBカラーコンバインの両方について、2

つのモードがある。1つは、1画素当たり1サイクルのバイブラインモードであり、もう1つは、1画素当たり2サイクルのバイブラインモードである。2サイクルモードにおいては、カラーコンバイナ508は、2つの線形補間の算術計算を実行できる。一般的には、第2サイクルは、テクスチャおよびシェーディング色変調を行うために使用され（すなわち、カラーコンバイナ508の操作は、一般的には、1サイクルモードにのみ使用される）、第1のサイクルは、別の線形補間計算に使用できる（例えば、2つのミップマップタイルからの2つのバイリニアフィルタしたテクセル間の詳細な補間レベル）。カラーコンバイナ508も、また、この実施例において、図42に示された「アルファのフィクスアップ」を実行する（図97の「キーGBのセット」コマンド参照）。

【0250】「ブレンダ」上述したように、ブレンダ510は、カラーコンバイナ508により提供されたコンバインされた画素値を取り、フレームバッファ118の画素に対して、それらをブレンドできる。透明性は、フレームバッファ色画素に対して、ブレンドすることにより達成される。多角形エッジアンチエイリアシングは、奥行（z）範囲に基づいて、条件付カラーブレンドを使用するブレンダ510によって部分的に実行される。ブレンダ510は、更に、フォグ操作を2サイクルモードで実行できる。

【0251】ブレンダ510は、異なる条件的カラーブレンドおよびZバッファ更新を行うことができ、従って、図43に示された多様なタイプの表面すべてを取り扱うことができる（すなわち、不透明な表面、デカル表面、透明な表面、および相互浸透表面）。

【0252】ブレンダ510の重要な特徴は、アンチエイリアシング処理に関係していることである。ブレンダ510は、奥行範囲に基づいて画素をフレームバッファ118Aに条件付きでブレンドし、書き込むものである（「dz」奥行限界フィールドを含む、Zバッファフォーマットの実施例を示す図46を参照）。任天堂株式会社およびシリコン・グラフィックス社によって出願した、同時係属中の特許出願（整理番号0569）、発明の名称「画素フラグメントの結合のためのシステムと方法」を参照のこと。

【0253】この実施例において、ビデオインタフェイス210は、フレームバッファの読み出し時間に空間フィルタを適用して、アンチエイリアシングシルエットエッジを生成するために、周辺背景色を説明する。アンチエイリアシング機構は、表面またはラインタイプにより分類された順序付けられたレンダリングを要求する。以下に、Zバッファアンチエイリアシングモード用のレンダリング順序および表面/ラインタイプを示す。

1. すべての不透明な表面がレンダリングされる。
2. すべての不透明なデカル表面がレンダリングされ

る。

3. すべての不透明な相互浸透表面がレンダリングされる。

4. すべての透明な表面およびラインが最後にレンダリングされる。

これらは、どのような順序でもレンダリングされるが、適正な奥行順序は、適正な透明度をあたえる。

【0254】モードブレンダ518は、プリミティブを定義する三角コマンド（図73参照）に特定された係数のグループにより、一部制御される。従って、プリミティブは、三角コマンドによって特定されたように、ZバッファモードまたはノンZバッファモードでレンダリングすることができる。加えて、「他のモードのセット」コマンド（図86、図87参照）は、「ブレンドマスク」および可能/不可能アンチエイリアシングを特定することのほかに、サイクル0およびサイクル1のブレンドモードワードを特定する。

【0255】ブレンダ510は、フォグカラーおよびブレンドカラーの2つの内部カラーレジスタを有している。これらの値は、それぞれ「フォグカラーのセット」および「ブレンドカラーのセット」コマンド（図90、図91参照）を用いてプログラム可能である。これらの値は、不変のフォグまたは透明性を有する幾何学のために使用できる。

【0256】ブレンダ510は、入来する画素アルファ値をプログラム可能なアルファソースと比較し、フレームバッファ118Aを条件付で更新させることができる。この特徴は、例えば、複雑でアウトラインされたビルボードタイプのオブジェクトを許容することができる。値に対するしきい値のほかに、この実施例のブレンダ510は、ディザされた値と比較して、ランダム化された微少な効果を与える。「他のモードのセット」コマンド（図86、図87）を参照。ブレンダ510は、1サイクルまたは2サイクルモードのいずれかで、フォグ操作を実行することもできる。ブレンダ510は、フォグおよびバイブラインカラーブレンドのために、ステップしたZ値をフォグ係数として使用する。

【0257】図44は、この実施例でブレンダ510が実行する全体の操作の実施例を示している。この特定の実施例において、ブレンダ510は、有効範囲エバリュエーター524が生成する有効範囲値が、ブレンドする量を特定することのできるモードで操作することが可能である。有効範囲エバリュエーター524は、（エッジウォーク504が提供した）現在の画素の有効範囲値を、フレームバッファ118A内の保存された有効範囲値と比較する。図45（カラーフレームバッファ118A内の各画素のために保存された色情報のフォーマットを示す）で示されるように、画素の色は、それぞれ5ビットの赤、緑および青の値および3ビット「有効範囲」によって表される。この「有効範囲」値は、現状のまま

で利用できるか、または画素アルファおよび／または有効範囲として使用するために、アルファ値によって乗算することができる(図86、図87の「他のモードのセット」コマンド参照)。「有効範囲」値は、画素のうちのどれだけが特定の表面により覆われるかを特定する。従って、エッジウォーク504が出力した有効範囲値は、完全にプリミティブの内部に存在している画素については1で、プリミティブの端にある画素については、1未満の値もある。この実施例においては、ブレンダ510は、アンチエリアジングのために有効範囲値を使用している。ブレンダ510がプリミティブエッジをブレンドする時は、そのプリミティブエッジが多数のプリミティブから形成されたオブジェクトの内部にあるのかどうか、または、そのエッジが代表のオブジェクトの外側端にあるのか分からない。この実施例において、この問題を解決するため、ビデオインタフェイス210が表示するためにフレームバッファ118Aを読み出す時に、不透明なエッジ値の最終的なブレンドは、表示するまで延期される。ビデオインタフェイス210は、この有効範囲値を用いて、画素色とフレームバッファ118A内の隣接画素の色との間を補間する。表示時にこのアンチエリアジングを達成するために、ブレンダ510は、フレームバッファ118a内の各画素に対する有効範囲値を維持しなければならず、それにより、特定の画素が多角形オブジェクトのシルエットエッジであるのか、または内部のエッジであるのかを、ビデオインタフェイス210が後に決定できるようになる。

【0258】メモリインタフェイス512およびZバッファリングメモリインタフェイス512は、表示プロセッサ500およびメインメモリ300の間のインタフェイスを提供する。メモリインタフェイス512は、主に通常の表示プロセッサ500の操作中に、カラーフレームバッファ118aおよびZバッファ118bにアクセスするために使用される。カラーフレームバッファ118aは、カラーテレビ画面60の各画素の色値を記憶する。画素フォーマットは、図45に示されている。Zバッファ118bは、カラーフレームバッファ118aに記憶される各色画素値の奥行値および奥行範囲値を保存する。Zバッファ値のフォーマットの実施例は、図46に示されている。Zバッファ118bは、主にブレンダ510により、新たにレンダリングされたプリミティブは、前にレンダリングされたプリミティブの前か後ろかを決定するために使用される(それより、隠れた表面除去を行う)。図46に示された「DZ」奥行範囲値は、隣接するテクセルは、同一のオブジェクト表面の一部であるか否かを決定する助けのために使用できる。

【0259】メモリインタフェイス512は、メインメモリ300に書き込み、メインメモリから読み取り、メインメモリ内の位置の読み取り、メインメモリ内の位置の修正および書き込み(RMW)をすることができる。

RMW操作につき、この実施例では、メモリインタフェイス512は、スパンのx、yの座標をエッジウォーク540が決定すると、直ちにフレームバッファ118aから画素の列をブレフェッチする。メモリインタフェイス512は、この画素のスパンまたは列を保存するのに使用される内部「スパンバッファ」512aを含んでいる。メモリインタフェイス512は、適切なブレフェッチ画素値を、スパンバッファ510aから、ブレンダ510に適切な時間で提供する、従って、メインメモリ300へのアクセスの数を少なくする。スパンバッファ512aは、表示プロセッサ500が、新しい画素値がブレンドされる度にメインプロセッサ300にアクセスする必要がないように、ブレンドされた(修正された)画素値を一時的に記憶するためにも使用される。一般に、メモリインタフェイス512は、画素の全体スパン値を、ブロックとして全部1度にメインメモリ300に書き込む。

【0260】メモリインタフェイス512は、幾つかのスパンバッファを保有するに十分なオンチップRAMを有している。しかし、連続した2つのスパンが、同一の画面領域で重複した場合には、問題を生じることがある。「他のモードのセット」コマンドにおける「アトミックスペース」パラメータ(図86、図87参照)は、メモリインタフェイス512に次のプリミティブを開始する前に、フレームバッファ118aに1プリミティブを書き込むことを強制する。それにより、プリミティブの最後のスパンがレンダリングされた後、サイクルが加わらないようにして、この潜在的問題を避けている。

【0261】奥行パラメータ526は、Zバッファ118bとともに動作して、隠された表面を除去し、透明値が適切にブレンドされることを保証する。奥行比較器526は、現在の画素のZまたは奥行値を、当該画面位置のZバッファ118aに現に存在するZ値と比較する。新規フレームをレンダリングする初期に、Zバッファ118bにあるすべての位置は、ビューアから最大の距離に好ましく初期化される(従って、いかなるオブジェクトもこの初期化値の「前面」でオープンする)。一般的に、表示プロセッサ500は、(新たな画素をフレームバッファ118aにブレンドする)、奥行比較器526は、現在の画素の奥行をZバッファ118bの位置に存在する奥行と比較する。古いZバッファ値が、以前に書き込まれた画素が、ビューアに新たな画素より「接近」していることを示す場合には、新たな画素(少なくとも不透明値については)は捨てられ、フレームバッファには書き込まれない。よって、隠された表面の除去が達成される。新たな画素が奥行比較器526により示されるように、古い画素に「接近」している場合には、新たな画素値は、(少なくとも不透明画素については)フレームバッファ118aの古い画素値と交替する。そして、Zバッファ118bの対応値は、同様に新たな画素のZ位

置とともに更新される(図47参照)。透明ブレンドは、Zバッファ値を更新しないで、ブレンドによって達成することができるが、それにも拘わらず、透明画素が不透明画素の「後ろ」の場合には、まず読み取りブレンドをしない。

【0262】[ビデオインタフェイス210]ビデオインタフェイス210は、フレームバッファ118からデータを読み取り、コンポジット、SビデオのRGBビデオ出力信号を発生する。この実施例では、ビデオインタフェイス210は、また、アンチエリアジング演算を実行し、更に、ディザリングノイズの混入により生じた切捨てを無くすフィルタリングを実行する。

【0263】この実施例では、ビデオインタフェイス210は、NTSCまたはPALモードのいずれかで動作し、高解像度および低解像度でフィルタリングを行って、あるいは行わずに、15ビットあるいは24ビットの色画素を表示できる。ビデオインタフェイス210は、また、小さい画像を全画面まで拡大することができる。ビデオインタフェイス210は、28の異なるビデオモードと、別の特別な特徴を提供する。

【0264】ビデオインタフェイス210は、カラーテレビ画面60を走査する電子ビームに同期してカラーフレームバッファ118aを読み取り、この実施例では、アナログビデオレベルに変換するために、各画素についてのRGB値をデジタル形式でビデオDAC144に与える。ビデオインタフェイス210は、有効範囲に基づいて不透明値に対してブレンド機能を実行し(これにより、アンチエリアジング機能を与える)、また、画面基準のディザリングにより生じたノイズのうちのあるものを除去するバックフィルタリング動作を実行する。

【0265】図48は、ビデオインタフェイス210のアーキテクチャのブロック図である。この実施例では、ビデオインタフェイス210は、DMAコントローラ900、バッファ902、制御論理904、アンチエリアジングフィルタ906a、906b、誤り訂正ブロック908aと908b、垂直補間器(フィルタ)910、水平補間器(フィルタ)912、「ランダム」関数発生器914、ガンマブロック916、およびバースドライバ918を備えている。

【0266】DMAコントローラ900は、コプロセッサバス214に接続されている。DMAコントローラ900は、メインプロセッサ100により特定されたメインメモリ300内の「原点」アドレスで始まるカラーフレームバッファ118aを読み取る(図50(b)参照)。DMAコントローラ900は、テレビ受像機58のライン走査動作と同期して、フレームバッファ118aから画素色および有効範囲値(図45参照)を連続的に読み取る。DMAコントローラ900により読み取られた画素値は、ビデオインタフェイス210の残りの部分により処理され、この実施例では、アナログ複合ビデオ

オ信号のNTSCあるいはPAL形式に変換するために、ビデオDAC144に出力される。

【0267】DMAコントローラ900は、この実施例では、メインメモリ300のフレームバッファ118aから読み取った色/有効範囲の値を一時記憶するために、RAMバッファ902に与える。この実施例では、バッファ902は、テレビジョンビデオの全ラインに対応する画素色値を記憶するわけではない。代わりに、バッファ902は、多数のブロックの画素データを記憶し、各ブロックは、ビデオラインの一部に対応する。バッファ902は、「ダブルバッファリング」を与える。すなわち、バッファ902は、あるライン部分をフィルタ906で利用できるようにするのに十分なバッファを備える一方で、他のバッファは、DMAコントローラ900により書き込まれる。

【0268】この実施例では、DMAコントローラ900は、カラーテレビ画面60上に表示されるべきビデオラインの幾つかの水平に配列された部分に対応する画素データのうちの幾つかにアクセスし、かつ、これらをバッファ902に記憶する。図49を参照すると、例示として、フレームバッファ118aが、テレビ画面上の画素に対応する列/行の順で組織されているものとして示されている(メインメモリ300に記憶されているフレームバッファは、実際には、画素色/有効範囲値の長い連続したリストとして記憶されていることがわかる)。この実施例では、DMAコントローラ900は、表示されるべきビデオの現在のラインnの特定のセグメントに対応する画素値のブロック(図49のフレームバッファ118aの上部斜線ブロック)を読み出し、かつ、「次の」ビデオラインn+1の(テレビ画面上に)水平に配列されたラインセグメントに対応する画素値(つまり、ラインnのすぐ下の「次の」ラインの部分を表す画素データの部分)を読み出す。この実施例では、DMAコントローラ900は、更に、フレームバッファからビデオラインn+2の水平に配列されたラインセグメントに対応する画素値のブロックを読み出す。

【0269】画素値のこれらのブロックは、それぞれバッファ902内に記憶される。フィルタ906a、906bは有効範囲値に基づいてフィルタリング/アンチエリアジング動作を実行し、現在のライン画素値を隣接画素値(つまり、カラーテレビ画面60上の表示位置に隣接する画素値)で補間する。フィルタ906a、906bにより実行されるアンチエリアジング・フィルタリング動作は、任天堂株式会社およびシリコン・グラフィックス社によって出願した、同時係属中の特許出願(整理番号0566)、発明の名称「シルエットエッジのアンチエリアジング」に記載されているものである。つまり、高隣接の3走査線は、フィルタ906により実行されるブレンド処理において、有効範囲値により色重みづけされている。このフィルタリング動作は、フレームバ

ッファ118a(この有効範囲値は、画素の何パーセントが多角形により覆われているかを示す)内に保持された画素有効範囲値を使用することにより、表面エッジにおいてより滑らかでよりジャグの少ない線を生じさせ、現在の画素値を発生するために、ブレンド処理における隣接画素値の影響に対応させて画素有効範囲値の影響を調整する。「ディボット」エラー修正ブロック908aおよび908bは、アンチエリアジング処理により導入された少量のアーチファクトについて、アンチエリアジングフィルタ906aおよび906bの出力を修正する。特に、シルエットエッジ上のあるいは隣接する画素すべてに対して、エラー修正ブロック908は、中央の画素に代えて表示されるべき色として3つの隣接画素の中央値をとる。このエラー修正は、ソフトウェアの制御下で使用可能にされ、あるいは使用禁止される(図50(a)参照)。このエラー修正は、デカル線レンダリングモードとは不十分にしか相互作用しないので、ビデオゲームのプログラマは、通常、このエラー修正を使用禁止することを望むだろう。

【0270】アンチエリアジングフィルタ906aおよび906bは、この実施例では、並列に動作し、フレームバッファ118aにより表された画像の2つの連続した線(ラインn、ラインn+1)の水平に配列された部分に対応する画素データブロックを生成する。これらの画素値は、垂直補間器910に与えられる。垂直補間器910は、2つの画像ライン間で線形補間を実行し、単一走査線の画像部分を生成する(図49参照)。補間器910は、飛越し走査表示におけるフリッカを低減するために、連続走査線間を補間する。例えば、補間器910は、先にあるいは次に連続する水平に配列された走査線部分の影響を取り込んで、連続したビデオ走査線間の遷移をより知覚できないようにして、これにより、フリッカを低減する。

【0271】更に、補間器910は、この実施例では、テレビ画面60上に表示されるラインの数が、フレームバッファ118aの画素情報により表されるラインの数とは異なることを可能にする垂直スケーリング機能を実行できる。この実施例では、フィルタ906は、フレームバッファ118aにより表される画像の連続したラインに対して画素データをリサンプリングすることにより、垂直方向の大きさをスケーリングする。これにより、テレビ画面60が異なるライン数をとることができる。このスケーリング動作(これはオフセット動作にも適合する)は、ビデオインタフェイスのYスケールレジスタ内の値により制御される(図52(n)参照)。

【0272】フレームバッファ118aのデジタル画像サイズに比例してテレビ画像をスケーリングするこの能力は、更にフレキシビリティを提供する。例えば、このスケーリング能力は、信号プロセッサ400および表示プロセッサ500が、フレームバッファ118内でより

小さいデジタル画像表示を生成することを可能にし、しかも、このより小さい画像がテレビ画面60の全体を満たすことも可能にする。小さいフレームバッファ118は、ラスターライズのためにより少ない時間を必要とし(つまり、表示プロセッサ500は、所与の多角形に対してより少ないスパンを処理し、かつ、スパン当たりより少ない画素を処理することが必要とされる)、また、記憶のためにより少ないメモリを必要とするだけであるので、このスケーリング能力は、性能を増すことができる。たとえ、より低い解像度の画像を犠牲にしても。

【0273】この実施例の垂直フィルタ910の出力は、表示されるべきビデオラインの一部に対して画素値を表す画素データのブロックである。図49に示されているように、画素値のこのブロックは、水平補間器912に与えられる。水平補間器912は、Xスケールレジスタに格納されている水平スケーリング係数に基づいて画素をリサンプリングするために、隣接画素値間で線形補間を行う(図52(m)参照)。このように、水平補間器912は、水平スケーリング能力を提供する。例えば、水平線に沿って少ない数のフレームバッファ値を多い数の画面画素に変換する。

【0274】水平補間器912の出力は、ガンマ補正回路916に供給される。ガンマ補正回路916は、線形RGB強度を、TVモニタのガンマ非線形を生成する複合ビデオ信号に適した非線形強度値に変換する。これは、線形色空間の平方根をとることに帰する。TVモニタは、これらの色値を2.2あるいは2.4の累乗まで効果的に立ちあげる。「ランダム」関数ブロック914は、「非ディザ」するために(つまり、表示プロセッサのディザリングブロック522により実行されるビット切捨てを補償するために)、解像度の付加ビットをR、GおよびBの各色値に導入する。図45に示されているように、この実施例では、フレームバッファ118の色画素フォーマットの一実施例は、メインメモリ300内に記憶スペースを保存するために、各R、G、Bの解像度の5ビットだけを与える。表示プロセッサのディザリングブロック522は、ブレンダ510により与えられる8ビットRGB色値を切り捨て、図45に示した圧縮表示を提供してもよい。ブロック914は、RGB値を圧縮解除することをこの切捨て処理に置き換えることができ、各R、GおよびBに256個の異なる表示色レベルを提供することができる。任天堂株式会社およびシリコン・グラフィックス社によって出願した、同時係属中の特許出願(整理番号0567)、発明の名称「ビット間引きされた画素の復元フィルタ」を参照されたい。このディザフィルタ動作は、ソフトウェアの制御下でオンおよびオフできる(図50(a)参照)。

【0275】[ビデオインタフェイスレジスタの実施例]同期信号発生、ビデオ再スケーリング、およびアンチエリアジングを含む全ての機能を制御するビデオイン

タフェイス210について、16個の制御レジスタがある。図50(a)～図52(p)は、メインプロセッサ100によりアクセスできるビデオインタフェイス210内の各種のレジスタを示している。

【0276】図50(a)は、ビデオインタフェイス制御レジスタ952を示している。メインプロセッサ100は、ビデオインタフェイス210の動作を制御するために、以下の値をレジスタ952へ書き込むことができる。

- ・ブランク（データなし、同期なし）、図45に示されたフォーマット（RGBの各々に5ビットおよび3ビット有効範囲値）、あるいは8/8/8/8（32ビット色値および8ビットの有効範囲）としての画素データサイズを特定する、タイプフィールド952a
- ・マハバンディングアーチファクトを除去するための7ビットの最終量子化の前に、ビデオ出力の最下位ビットに対してランダムノイズを付加することをオンおよびオフする、ガンマディザ使用可能フィールド952b
- ・ガンマ修正をオンおよびオフする、ガンマ使用可能フィールド952c
- ・前述のデボットエラー修正をオンおよびオフする、デボット使用可能フィールド952d
- ・内部クロックをオンまたはオフする、ビデオバスクロック使用可能フィールド952e
- ・飛越し走査をオンおよびオフする、インタレースフィールド952f
- ・テストモードフィールド952g、
- ・アンチエリアジングモードオン/オフフィールド952h
- ・診断フィールド952i
- ・画素アドバンスフィールド952j
- ・ディザフィルタ使用可能フィールド952k

【0277】図50(b)は、読み出しのために、フレームバッファ118aのメインメモリ300のアドレススタートを特定するビデオインタフェイス原点レジスタ954を示している。この例では、メインプロセッサ100は、ビデオインタフェイス210がメインメモリ300の新しい領域から読み出す（例えば、ダブルバッファされたフレームバッファ118の残りの半分を読み出す）度に、このレジスタ954を明確にセットすることを必要とする。

【0278】図50(c)は、各水平ラインにおける画素の数を特定するためにセットされるビデオインタフェイスライン幅レジスタ956を示している。図50(d)は、コプロセッサ200が特定された垂直ラインあるいはハーフラインにおいて、フレーム毎にメインプロセッサに割り込むように、メインプロセッサ100が特定の垂直線数でセットできるビデオインタフェイス垂直割り込みレジスタ958を示している。図50(e)は、メインプロセッサ100により読み取られた時に、

現在の垂直ラインを特定し、かつ、メインプロセッサ100により書き込まれた時に、垂直ライン割り込みをクリアするビデオインタフェイスカレントラインレジスタ960を示している。

【0279】図51(g)～図52(1)に示されたレジスタ962～972は、詳細な複合ビデオタイミングパラメータを特定するために、メインプロセッサ100により使用される。例えば、

・図51(f)は、水平同期パルス幅、カラーバースト幅、垂直同期パルス幅、およびカラーバースト開始タイミングを特定するために、メインプロセッサ100が書き込むことができる垂直インタフェイスタイミングレジスタ962を示している。

・図51(g)は、フィールド毎の垂直ハーフラインの数を特定するために、メインプロセッサ100が書き込むビデオインタフェイス垂直同期レジスタ964を示している。

・図51(h)は、PAL方式のためのラインおよび水平「リープパターン」の全持続時間を特定するために、メインプロセッサ100が書き込むビデオインタフェイス水平同期レジスタ965を示している。

・図51(i)は、PAL方式のための2つの交番水平同期リープパラメータを特定するビデオインタフェイス水平同期リープレジスタ966を示している。

・図51(j)、図51(k)に示されたビデオインタフェイス水平ビデオレジスタおよび垂直ビデオレジスタ968、970は、それぞれ水平同期および垂直同期に対して水平および垂直ビデオ開始および終了時を特定するために使用される。

・図52(1)に示された垂直インタフェイスされた垂直バーストレジスタ972は、カラーバースト開始および終了タイミングを特定する。

【0280】レジスタ962～972にプログラムできるタイミングパラメータは、異なる種類のテレビ受像機58に互換性を与えるために使用される。例えば、米国における大半のテレビ受像機58は、NTSCとして知られている複合ビデオフォーマットを使用しており、一方、欧州の大半のテレビ受像機58は、PALとして知られている複合ビデオフォーマットを使用している。これらのフォーマットは、詳しいタイミングパラメータ

（例えば、信号パターン内の垂直ブランキング幅および位置、水平同期パルス幅、カラーバースト信号パルス幅等）において異なる。レジスタ962～972は、これらの複合ビデオタイミングパラメータを制御し、メインプロセッサ100上のソフトウェアの実行によりプログラムできるので、ビデオゲームプログラム108のプログラムは、適当な値をレジスタ962～972に書き込むビデオゲームプログラム内に適当な命令を含ませることにより、そのプログラムを（ユーザが選択したように）NTSC互換、PAL互換、あるいは両方互換にす

ることができる。この例では、コプロセッサ200は、NTSC標準テレビ受像機58、PAL標準テレビ受像機58に互換でき、そしてレジスタ962~972の内容によって特定された範囲内で、これら以外のビデオフォーマットによっても互換できる。

【0281】垂直インタフェイスxおよびysケールレジスタ974、976（それぞれ、図52（m）、図52（n）参照）は、前述したように、水平および垂直スケリングにおいて、xおよびysケールのアップおよび副画素オフセットパラメータを特定する。図52（o）および図52（p）は、診断のためのビデオインタフェイステストデータおよびアドレスレジスタ978、980を示している。

【0282】[メモリコントローラ/インタフェイス212] 前述したように、コプロセッサメモリインタフェイス212は、メインメモリ300をコプロセッサ内部バス214にインタフェイスで接続する。この実施例では、メインメモリ300は、9ビット幅バスを介してアクセスされ、そしてメモリインタフェイス212が対応できるタスクのひとつは、コプロセッサ200によってもっと便利に取り扱うことができるように、連続した9ビットワードをバッファすることである。図53は、メモリコントローラ/インタフェイス212の全体アーキテクチャの実施例を示す図である。

【0283】この実施例では、メモリインタフェイス/コントローラ212は、一対のレジスタ/バッファ1000、1002、制御ブロック1004、およびRAMコントローラブロック212bを有している。RAMコントローラブロック212bは、メインメモリ300を制御するためにRambus社により設計され、かつ特定されたRAM制御回路を備えている。レジスタ1000、1002は、それぞれ出力データおよび入力データをラッチするために使用される。制御ブロック1004は、メモリインタフェイス212の動作を制御する。

【0284】[メモリコントローラ/インタフェイスレジスタの例] 図54（a）~図55（h）は、メモリインタフェイス212を制御するために、メインプロセッサ100により使用される制御レジスタの例を示している。図54（a）は、動作モードを特定しかつ送信あるいは受信のどちらがアクティブにあるかを特定する読み取り/書き込みモードレジスタを示している（1052）。図54（b）は、現在の制御入力および現在の制御使用可能を特定するコンフィギュレーションレジスタ1054の構成を示している。図54（c）は、書き込み専用の現在のモードレジスタ1056を表しており、このレジスタへの書き込みが現在の制御レジスタを更新する。図54（d）は、受信あるいは送信を選択するために使用される選択レジスタ1058を示している。図54（e）は、DMA待ち時間/オーバーラップを特定するために使用される待ち時間レジスタ1060を示して

いる。図54（f）は、クリーンおよびダーティリフレッシュ遅延を特定し、現在のリフレッシュバンクを示し、リフレッシュが使用可能か否かを示し、リフレッシュが最適か否かを示し、またリフレッシュマルチバンク装置を特定するフィールドを有するというリフレッシュレジスタ1062を示している。図55（g）は、リードモードでNACK、ACKおよびオーバーランエラーを示し、かつメインプロセッサ100により書き込まれた時に、全てのエラービットをクリアするエラーレジスタを示している。図55（h）は、読み取られた時にバンクおよび現在のバンクのダーティビットを示し、また、書き込まれた時にバリッドをクリアし現在のバンクのダーティビットをセットする、バンク状態レジスタ1066を示している。

【0285】[CPUインタフェイス] 図56は、この実施例における、コプロセッサのCPUインタフェイス202のブロック図を示している。CPUインタフェイス202は、FIFOバッファ1102および制御ブロック1104から成っている。FIFOバッファ1102は、CPUSysAD多重化アドレス/データバス102aと、コプロセッサ多重化アドレス/データバス214Dとの間に、双方向性バッファを与える。制御ブロック1104は、メインプロセッサ100により指定されたアドレスを受信し、それらをコプロセッサアドレスバス214C上に出力する。制御ブロック1104は、コプロセッサ200の他の部分から割り込み信号を受け取り、また、SysCMDバス102bを介して、メインプロセッサ100からコマンド制御信号を受け取る。

【0286】[CPUインタフェイスレジスタの実施例] 図57（a）~図57（d）は、この実施例では、CPUインタフェイス303内に含まれたレジスタを示している。図57（a）は、メインプロセッサ100がレジスタに書き込んだ時にコプロセッサ200を制御し、かつ、メインプロセッサ100がレジスタから読み取った時にコプロセッサ200の全状態を示す、CPUインタフェイス状態/制御レジスタ1152を示している。メインプロセッサ100は、レジスタ1152に書き込んで初期設定コード長を特定し、初期設定モードをセットあるいはクリアし、内部コプロセッサバステストモードをセットあるいはクリアし、信号プロセッサ400の割り込みをクリアし、およびメインメモリレジスタモードをセットあるいはクリアする。メインプロセッサ100は、このレジスタ1152から読み取った時に、初期設定コード長、初期設定モード、内部コプロセッサバステストモード、およびコプロセッサ200がメインメモリレジスタモードで動作しているか否かを決定できる。

【0287】図57（b）は、メインプロセッサ100が、コプロセッサ200内の種々の要素に関するバージョン情報を決定するために読み取るバージョンレジスタ

10

20

30

40

50

1154を示している。

【0288】図57(c)は、コプロセッサ200から受け取った割り込み源を決定するために、メインプロセッサ100が読み取る割り込みレジスタ1156を示している。この実施例では、コプロセッサ200とメインプロセッサ100とを接続するシングルラインが、割り込み目的に使用される。コプロセッサの割り込みを受け取ると、メインプロセッサ100は、(割り込みベクトルを含んでいる)割り込みレジスタを読んで、コプロセッサ200内のどの要素(つまり、信号プロセッサ400、シリアルインタフェイス204、オーディオインタフェイス208、ビデオインタフェイス210、パラレルインタフェイス206、あるいは表示プロセッサ500)が割り込みを発生したかを確認する。図57(d)は、メインプロセッサ100が、割り込みレジスタ1156で特定された割り込みのいずれかに対して、割り込みマスクをセットあるいはクリアするために書き込み、かつ、割り込みがマスクされているか否かを決定するために読み取られる割り込みマスクレジスタ1158を示している。

【0289】[オーディオインタフェイス]図58は、この実施例における、オーディオインタフェイス208のアーキテクチャの全体ブロック図を示している。オーディオインタフェイス208は、DMA論理1200、状態マシン/コントローラ1202、オーディオクロック発生器1204、オーディオデータバッファ1206、および並直列変換器1208を備えている。この実施例では、DMA論理1200は、メインメモリ300内のオーディオバッファ114からデジタルオーディオサンプルデータを取り出す。DMA論理1200は、一度に8バイトでオーディオサンプルデータをオーディオデータバッファ1206に書き込む。FIFO内に複数のオーディオデータバッファ1206が設けられており、そのためDMA論理1200が、あるオーディオサンプルデータを先取りでき、一方、並直列変換器1208が、他の先に取出されかつバッファされたオーディオサンプルデータを並直列変換する。このように、オーディオデータバッファ1206は、DMA論理1200によって読み取られるブロック間に、並直列変換器1208を供給するのに十分なデータを記憶する。並直列変換器1208の出力速度は、比較的遅いので(例えば、50kHzで4バイトのオーダー)、単一の64ビットバッファ1206bは、リアルタイムオーディオ出力として比較的長い時間持続するために十分なデジタル化オーディオサンプルを記憶できる。

【0290】前述のように、並直列変換器は、オーディオバッファ1206のパラレルな内容をシリアルフォーマットに変換し、得られたシリアルデジタルオーディオデータのストリームを、オーディオDAC140へ通信するためバス209に出力する。この実施例では、デジ

タルオーディオバス209は、左チャンネルデータと右チャンネルデータとの間で多重化されている単一シリアルデータライン209aを有している。この実施例では、並直列変換器1208は、チャンネル交互に各ステレオチャンネルへ16ビット長ワードを出力する。並直列変換器1208の出力ビット速度は、オーディオクロック発生器1204により特定される。オーディオクロック発生器1204は、オーディオDAC140を並直列変換器1208の出力ビット速度に同期させるために、オーディオクロック出力を209b上に発生し、また、現在の並直列変換器出力1208の出力が、左ステレオチャンネルに向けたものかあるいは右ステレオチャンネルに向けたものかを特定するため、オーディオL/Rクロックをライン209c上に発生する。

【0291】図58は、オーディオインタフェイス208を制御するために使用される多数のレジスタおよびカウンタを示している。DMAコントローラ1200は、アドレスレジスタ1210から開始メインメモリアドレスを受け取る。メインプロセッサ100は、アドレスレジスタ1210に書き込み(図59(a)参照)、オーディオインタフェイス208をメインメモリ300内の記憶場所に向けて、再生されるべき現在のオーディオをオーディオバッファ114に与える。カウンタ1212は、DMAコントロール1200による各取り出し毎にこのアドレスを増分し、これにより全オーディオバッファ114を介して、DMAコントローラ1200を順序づけする。メインプロセッサ100は、オーディオバッファ114の長さを転送長レジスタ1214に書き込む(図59(b)参照)。長さレジスタ1214に関連する別のカウンタ1216は、オーディオバッファ114の長さに対応する適当な数の制御状態によって、状態マシン1202を順序づけする。状態マシン1202は、互いにオーディオインタフェイス208の他の部分の動作の同期をとる制御信号を発生する。この実施例では、メインプロセッサ100は、オーディオインタフェイス208を使用可能にして、DMA使用可能レジスタ1216の記憶場所(図58に図示せず、図59(c)を参照)に書き込むことにより、メインメモリ300からのデータの取り出しを開始する。メインプロセッサ100は、また、オーディオインタフェイス状態レジスタ1218を読み取ることによりオーディオインタフェイス208の状態を決定する(図58に図示せず、図59(d)を参照)。この実施例では、状態マシン1202は、長さレジスタ1214により特定されたオーディオバッファ114の終了に達した時に、メインプロセッサ割り込みを発生する。また、メインプロセッサ100は、状態レジスタ1218の記憶場所へ書き込むことにより、この割り込みをクリアできる(図59(d)参照)。

【0292】この実施例では、メインプロセッサ100

は、また、オーディオクロック発生器1204により発生されたクロック信号の速度を制御できる。メインプロセッサ100は、オーディオ速度レジスタ1218、1220に書き込むことによりこれらの速度をプログラムできる(図59(e)、図59(f)参照)。カウンタ1222は、メインプロセッサ100がオーディオ速度レジスタ1218、1220に書き込んだ速度値に基づいて、プログラム可能な分割機能を与える。

【0293】[シリアルインタフェイス]図60は、この実施例における、シリアルインタフェイス204の全10
体高レベルブロック図を示している。

【0294】この実施例では、シリアルインタフェイス204は、コプロセッサ200とシリアル周辺インタフェイス138との間で、データのブロックを移動する。シリアルインタフェイス204は、シリアル周辺インタフェイス138から64バイトのデータブロックを読み取り、これをメインメモリ300内の特定された記憶位置に転送するか、あるいはメインメモリ300に記憶されている64バイトブロックのデータを読み取り、これを直列的にシリアル周辺インタフェイスに転送する。この実施例では、シリアルインタフェイス204は、主にダイレクトメモリアクセス論理1300、制御論理1302、およびパラレル/シリアル変換器1304を有している。パラレル/シリアル変換器1304は、この実施例では、読取りデータ/肯定応答バス205aを介して、シリアル周辺インタフェイス138により送られたシリアルデータをラッチ1308に適用するため、パラレルデータに変換するシフトレジスタを有している。ラッチ1308の内容は、メインメモリ300への書き込みのために、コプロセッサデータバス214dに供給される。代わりに、パラレル-シリアル変換モードで、シフトレジスタ1304は、ラッチ1310を介してコプロセッサデータバス214dからパラレルデータを受け取り、このデータをコマンドおよび書き込みデータバス205bを経由して、シリアル周辺インタフェイス138へ転送されるシリアルに変換する。

【0295】メインプロセッサ100は、アドレスをアドレスレジスタ1312に書き込むことにより、シリアルインタフェイス204が読み取り、かつ、書き込むメインメモリ300内のアドレスを特定する(図61(a)参照)。アドレスレジスタ1312の内容は、DMAアドレスカウンタ1314にロードされるべきメインメモリ300のアドレスを特定する。アドレスレジスタ1312の内容の一部分は、また、シリアル周辺インタフェイス138内の「アドレス」情報を特定するためにも使用される。このシリアル周辺インタフェイスの「アドレス」情報は、ラッチ1316にロードされ、その内容がシリアル周辺インタフェイスに送るためにシフトレジスタ1304に与えられる。このシリアル周辺インタフェイスの「アドレス」情報は、例えば、シリアル
50

周辺インタフェイス138内の記憶場所(つまり、ブートROM記憶場所158、RAMバッファ、あるいは状態レジスタ)を特定するために使用される。

【0296】この実施例では、シリアルインタフェイス204は、シフトレジスタ1304のパラレル出力を、レジスタ1308、マルチプレクサ1318、およびラッチ1320を介してコプロセッサアドレスバス214c上に出力する能力を備えている。図61(b)、図61(c)に示されたように、この実施例では、メインプロセッサ100は、記憶場所1322あるいは1324に書き込むことによりシリアル転送の方向を特定する。記憶場所1322への書き込みは、シリアルインタフェイス204に64バイトデータブロックをシリアル周辺インタフェイス138から読み取らせ、かつ、このデータブロックをアドレスレジスタ1312により特定されたメインメモリ300の記憶場所に書き込ませる。メインプロセッサ100による記憶場所1324への書き込みは、シリアルインタフェイス204に64バイトのデータブロックをアドレスレジスタ1312により特定されたメインメモリ300の記憶場所から読み取らせ、かつ、このデータブロックをシリアル形式でシリアル周辺インタフェイス138に書き込ませる。

【0297】図61(d)は、シリアルインタフェイス状態レジスタ1326を示している。メインプロセッサ100は、シリアルインタフェイス204の状態(すなわち、シリアルインタフェイスがDMAあるいはI/O動作でビジーであるか否か(それぞれ、フィールド1328(1)、1328(2))、DMAエラーがあったか否か(フィールド1328(3))、あるいはシリアルインタフェイスがメインプロセッサの割り込みを発生させたか否か(フィールド1328(4)))を決定するために、状態レジスタ1326を読み取ることができる。シリアルインタフェイス204は、シリアル周辺インタフェイス138へのデータ転送、または、シリアル周辺インタフェイス138からのデータ転送を完了する度に、メインプロセッサの割り込みを発生する。メインプロセッサ100は、レジスタ1326へ書き込むことにより、シリアルインタフェイスの割り込みをクリアできる。

【0298】[パラレル周辺インタフェイス]図62は、パラレル周辺インタフェイス206のブロック図の実施例を示している。この実施例では、パラレルインタフェイス206は、データのブロックをメインメモリ300と記憶装置54との間で転送する。前述の記憶装置54は、パラレルバス104に接続された読み取り専用メモリ76を1つだけ備えているが、システム50は、コネクタ154に接続する周辺装置の異なる構成にも適合できる。例えば、2つの異なる形式の周辺装置(例えば、ROMおよびRAM)が周辺コネクタ154に接続できる。周辺インタフェイス206は、書き込み間で手

間のかかる再構成を必要とせずに、同じパラレルバス104に接続された2つの異なる形式の周辺装置間の通信をサポートするように設計される。

【0299】このような周辺装置のあるものは読み取り専用であり（例えば、ROM76）、周辺装置の別のものは読み書きでき（例えば、ランダムアクセスメモリあるいはモデム）、また周辺装置の更に別のものは書き込み専用である。周辺インタフェイス206は、コネクタ154とメインメモリ300との間のパラレルバス104を介した双方向性のパラレル転送をサポートする。

【0300】この実施例では、パラレル周辺インタフェイス206は、DAMコントローラ1400、制御/レジスタブロック1402、およびレジスタファイル1404を有している。レジスタファイル1404は、コネクタ154に接続された周辺装置とメインメモリ300内の記憶場所のブロックとの間で、周辺インタフェイス206により転送されるデータブロックをバッファする。この実施例では、レジスタファイル1404は、16個の64ビットワードを記憶する小さいRAMを備えている。レジスタファイル1404は、FIFOとして動作し、制御/レジスタブロック1402によりアドレス指定される。レジスタファイル1404の出力は、マルチプレクサによって16ビット部分に多重化される。これらの16ビット幅値は、多重化アドレス/データバス104adを介してコネクタ154に接続された周辺装置へ供給するために、ラッチ1408によりラッチされる。多重化アドレス/データバス104adを介して周辺装置から読み取られたデータは、（16ビット読み取り値を64ビットワードの適当な4分の1に配置するマルチプレクサ1412を介して）レジスタファイル1404に供給される前に一時的にラッチ1410に記憶される。マルチプレクサ1412は、ラッチ1414を介してコプロセッサデータバス214dからデータを受け取り、この受け取ったデータを記憶するためにレジスタファイル1404に送る。レジスタファイル1404の出力は、ラッチ1416を介してコプロセッサデータバス214dにも接続できる。この実施例では、レジスタファイル1404の出力は、マルチプレクサ1418およびラッチ1420を介してコプロセッサのアドレスバス214cにも接続されている。

【0301】メインプロセッサ100は、パラメータを制御/レジスタブロック1402に書き込むことにより、周辺インタフェイス206により実行されるDAM転送のパラメータを制御する。例えば、メインプロセッサ100は、開始メインメモリアドレスをDRAMアドレスレジスタ1422に書き込むことができ（図63（a）参照）、また周辺バスアドレス開始アドレスを周辺バスレジスタ1424に書き込むことにより、コネクタ154に接続された周辺装置の開始アドレス空間を書き込むことができる（図63（b）参照）。この実施例

では、メインプロセッサ100は、それぞれ図63（c）、図63（d）に示されたレジスタ1426、1428の一方に書き込むことにより、転送の長さおよび方向を特定する。図63（c）に示す読み取り長さレジスタ1426への書き込みは、一方の方向へ転送するようにパラレル周辺インタフェイス206を制御し、一方、図63（d）に示すレジスタ1428への長さ値の書き込みは、周辺装置に他方の方向への転送を行わせる。この実施例では、メインプロセッサ100は、状態レジスタの記憶場所1430（R）から読み出すことにより、周辺インタフェイス206の状態を読み取ることができる（図63（b）参照）。この状態レジスタ1430（R）は、進行中のDMA転送（フィールド1432（1））、処理中のI/O動作（フィールド1432（#））、エラー状態（フィールド1432（3））を示すフィールド1432を備えている。同じレジスタ1430（w）の記憶場所に書き込むことにより、メインプロセッサ100は、要求された転送を完了した時に発生する割り込み周辺インタフェイス206をクリアできる。状態レジスタ記憶場所1430（w）への書き込みは、またメインプロセッサ100が進行中の転送をクリアし割り込み、そして中止できるようにする（図63（a）、フィールド1434（1））。

【0302】図64（f）、図64（g）、図64（h）、図64（i）は、周辺インタフェイスバス104のタイミングおよび他のパラメータを制御するために、メインプロセッサ100が書き込むことができる別のレジスタを示している。これらのレジスタは、メインプロセッサ100が、全てゲームプログラム108内のソフトウェアの制御下で、特定の型式の周辺装置へのバス104を構成することを可能にする。この実施例では、周辺インタフェイス44は、図64（f）～図64（i）に示されたレジスタ1436、1438、1440、および1442の対のセットをサポートしている。これにより、メインプロセッサ100が異なる装置へのアクセスを要求する都度、メインプロセッサ100に構成レジスタを再書き込みすることを要求せずに、異なる周辺バス104のプロトコルが、バスに同時に接続された異なる周辺装置に使用できるようになる。この実施例では、パラレル周辺インタフェイス206が、16ビット周辺アドレス空間内の「領域1」のアドレス空間をアクセスする時にはいつでも、構成レジスタ1436、1438、1440、および1442の一方の組がバス104のプロトコル構成するように使用される。一方、周辺インタフェイスが周辺バスアドレス空間内の「領域2」のアドレス空間をアクセスする時にはいつでも、レジスタパラメータの他のセットが使用される（図10のメモリマップ参照）。これら2セットのレジスタにより特定される構成は、メインプロセッサ100により適当な領域へ書き込むことにより簡単に呼び出すことができ

る。

【0303】図63(a)～図64(i)に示された制御レジスタの種々のものは、この実施例では、図62の制御／レジスタブロック1402内に配置されている。レジスタ1436、1438、1442に記憶された構成値は、この実施例では、制御／レジスタブロック1402が、バス制御ライン104C上に発生するアクセス制御信号のタイミングを制御するために使用される。ラッチ1434は、制御／レジスタブロック1402へ供給するコプロセッサのアドレスバス214C上のアドレスを、一時的にラッチするために(例えば、種々のレジスタ間を選択するために)使用される。この実施例では、制御／レジスタブロック1402は、DMAアドレスを自動的に増分するための適当なカウンタ等を備えている。

【0304】本発明は、現段階において、最も実用的かつ好ましい実施態様と思われるものに関して記述されたものであるが、本発明は、公開された実施形態に限定されるものではなく、上記特許請求の範囲の精神と範囲において包含される種々の修正および均等範囲に属する変形や変更を含むことを意図するものである。

【図面の簡単な説明】

【図1】3D画像およびデジタル処理ステレオ音響を生成する全体ビデオゲームシステムを示している。

【図2】図1のシステムを用いて実現できる3D画面効果の実施例を示している。

【図3】図1のシステムを用いて実現できる3D画面効果の実施例を示している。

【図4】全体ビデオゲームシステムの主要要素の一実施例を示している。

【図5】全体ビデオゲームシステムの主要処理動作の一実施例を示している。

【図6】ビデオゲームシステムの全体動作の一実施例を示している。

【図7】グラフィック画像を生成するためにビデオゲームシステムにより実行される全ステップの一実施例を示している。

【図8】詳細な全体システムアーキテクチャの一実施例を示している。

【図9】メインプロセッサの初期化ルーチンの一実施例を示している。

【図10】メインプロセッサのメモリマップの一実施例を示している。

【図11】コプロセッサの内部アーキテクチャの一実施例を示している。

【図12】コプロセッサの内部バスのアーキテクチャの一実施例を示している。

【図13】信号プロセッサの内部アーキテクチャの一実施例を示している。

【図14】(a)は、信号プロセッサのインストラクシ

ョンフォーマットの一実施例を示しており、(b)は、図13に示されたベクトルユニットによる処理のための当該(a)のソース、あるいは行先フィールドのスライシングの一実施例を示しており、(c)は、信号プロセッサのベクトルユニットの例により、実行される追加の動作の一実施例を示している。

【図15】信号プロセッサのレジスタの一実施例を示している。

【図16】信号プロセッサのレジスタの一実施例を示している。

【図17】グラフィック表示リストおよびオーディオ再生リストを含む階層タスクリストの一実施例を示している。

【図18】マイクロコードロードルーチンの一実施例を示している。

【図19】例示している簡単な信号プロセッサの表示リストの処理の一実施例を示している。

【図20】信号プロセッサのグラフィックマイクロコードの制御ステップシーケンスの一実施例を示している。

【図21】(a)は、倍精度表現の一実施例を示しており、(b)は、マトリックスフォーマットの一実施例を示している。

【図22】(a)は、信号プロセッサの頂点バッファフォーマットの一実施例を示しており、(b)は、頂点データの定義の一実施例を示している。

【図23】信号プロセッサのセグメントアドレス指定の構成の一実施例を示している。

【図24】オーディオソフトウェアのアーキテクチャの一実施例を示している。

【図25】簡単な信号プロセッサの再生リストの処理の一実施例を示している。

【図26】信号プロセッサのオーディオマイクロコードの制御ステップシーケンスの一実施例を示している。

【図27】信号プロセッサのオーディオ処理構造の一実施例を示している。

【図28】表示プロセッサの全体処理ステップの一実施例を示している。

【図29】表示プロセッサのパイプライン構造の一実施例を示している。

【図30】表示プロセッサのアーキテクチャの一実施例を示している。

【図31】表示プロセッサのレジスタの一実施例を示している。

【図32】表示プロセッサのレジスタの一実施例を示している。

【図33】表示プロセッサのレジスタの一実施例を示している。

【図34】表示プロセッサのレジスタの一実施例を示している。

【図35】テクスチャメモリのタイル記述子の構成の

一実施例を示している。

【図36】テクスチャユニット処理の一実施例を示している。

【図37】テクスチャ座標ユニットおよびテクスチャメモリユニットのアーキテクチャの一実施例を示している。

【図38】テクスチャメモリのカラーインデックスモードのルックアップの一実施例を示している。

【図39】カラーインデックステクスチャを記憶するためのテクスチャメモリのより詳細な使用の一実施例を示している。

【図40】カラーコンバイナの動作の一実施例を示している。

【図41】アルファコンバイナの動作の一実施例を示している。

【図42】アルファフィックスアップの動作の一実施例を示している。

【図43】異なるタイプのプリミティブのブレンドの一実施例を示している。

【図44】ブレンダの動作の一実施例を示している。

【図45】色画素のフォーマットの一実施例を示している。

【図46】奥行画素のフォーマットの一実施例を示している。

【図47】書き込み可能生成処理の一実施例を示している。

【図48】ビデオインタフェイスのアーキテクチャの一実施例を示している。

【図49】ビデオインタフェイスの動作シーケンスの一実施例を示している。

【図50】ビデオインタフェイスの制御レジスタの一実施例を示している。

【図51】ビデオインタフェイスの制御レジスタの一実施例を示している。

【図52】ビデオインタフェイスの制御レジスタの一実施例を示している。

【図53】メインメモリインタフェイスのアーキテクチャの一実施例を示している。

【図54】メモリインタフェイスの制御レジスタの一実施例を示している。

【図55】メモリインタフェイスの制御レジスタの一実施例を示している。

【図56】メインプロセッサのインタフェイスのアーキテクチャの一実施例を示している。

【図57】メモリプロセッサのインタフェイスのレジスタの一実施例を示している。

【図58】オーディオインタフェイスのアーキテクチャの一実施例を示している。

【図59】オーディオインタフェイスのレジスタの一実施例を示している。

【図60】シリアルインタフェイスのアーキテクチャの一実施例を示している。

【図61】シリアルインタフェイスのレジスタの一実施例を示している。

【図62】周辺インタフェイスのアーキテクチャの一実施例を示している。

【図63】周辺インタフェイスの制御/状態レジスタの一実施例を示している。

【図64】周辺インタフェイスの制御/状態レジスタの一実施例を示している。

【図65】表示プロセッサにおけるグラフィックス表示コマンドの「カラー画像のセット」のフォーマットおよび関連ファンクションを示している。

【図66】表示プロセッサにおけるグラフィックス表示コマンドの「テクスチャ画像のセット」のフォーマットおよび関連ファンクションを示している。

【図67】表示プロセッサにおけるグラフィックス表示コマンドの「Z画像のセット」のフォーマットおよび関連ファンクションを示している。

【図68】表示プロセッサにおけるグラフィックス表示コマンドの「タイルのセット」のフォーマットおよび関連ファンクションを示している。

【図69】表示プロセッサにおけるグラフィックス表示コマンドの「タイルのロード」のフォーマットおよび関連ファンクションを示している。

【図70】表示プロセッサにおけるグラフィックス表示コマンドの「ブロックのロード」のフォーマットおよび関連ファンクションを示している。

【図71】表示プロセッサにおけるグラフィックス表示コマンドの「タイルサイズのセット」のフォーマットおよび関連ファンクションを示している。

【図72】表示プロセッサにおけるグラフィックス表示コマンドの「タルトのロード」のフォーマットおよび関連ファンクションを示している。

【図73】三角コマンドの種々のタイプを示している。

【図74】表示プロセッサにおけるグラフィックス表示コマンドの「エッジの係数」のフォーマットおよび関連ファンクションを示している。

【図75】表示プロセッサにおけるグラフィックス表示コマンドの「エッジの係数」のフォーマットおよび関連ファンクションを示している。

【図76】表示プロセッサにおけるグラフィックス表示コマンドの「エッジの係数」のフォーマットおよび関連ファンクションを示している。

【図77】表示プロセッサにおけるグラフィックス表示コマンドの「シェードの係数」のフォーマットおよび関連ファンクションを示している。

【図78】表示プロセッサにおけるグラフィックス表示コマンドの「シェードの係数」のフォーマットおよび関連ファンクションを示している。

* コマンドの「フィルカラーのセット」のフォーマットおよび関連ファンクションを示している。

【図93】表示プロセッサにおけるグラフィックス表示コマンドの「プリミティブ奥行きのセット」のフォーマットおよび関連ファンクションを示している。

【図94】表示プロセッサにおけるグラフィックス表示コマンドの「シザーのセット」のフォーマットおよび関連ファンクションを示している。

【図 95】表示プロセッサにおけるグラフィックス表示
10 コマンドの「変換のセット」のフォーマットおよび関連
ファンクションを示している。

【図96】表示プロセッサにおけるグラフィックス表示コマンドの「キーRのセット」のフォーマットおよび関連ファンクションを示している。

【図97】表示プロセッサにおけるグラフィックス表示
コマンドの「キーGBのセット」のフォーマットおよび
関連ファンクションを示している。

20 【図98】表示プロセッサにおけるグラフィックス表示
コマンドの「フルシンク」のフォーマットおよび関連フ
ァンクションを示している。

【図99】表示プロセッサにおけるグラフィックス表示コマンドの「ロードシンク」のフォーマットおよび関連ファンクションを示している。

【図100】表示プロセッサにおけるグラフィックス表示コマンドの「パイプシンク」のフォーマットおよび関連ファンクションを示している。

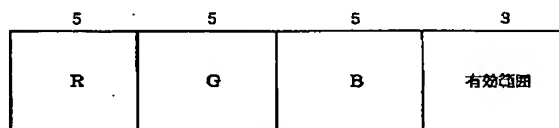
【図101】表示プロセッサにおけるグラフィックス表示コマンドの「タイルシンク」のフォーマットおよび関連ファンクションを示している。

30 【図102】表示プロセッサにおけるグラフィックス表示コマンドの「操作なし」のフォーマットおよび関連ファンクションを示している。

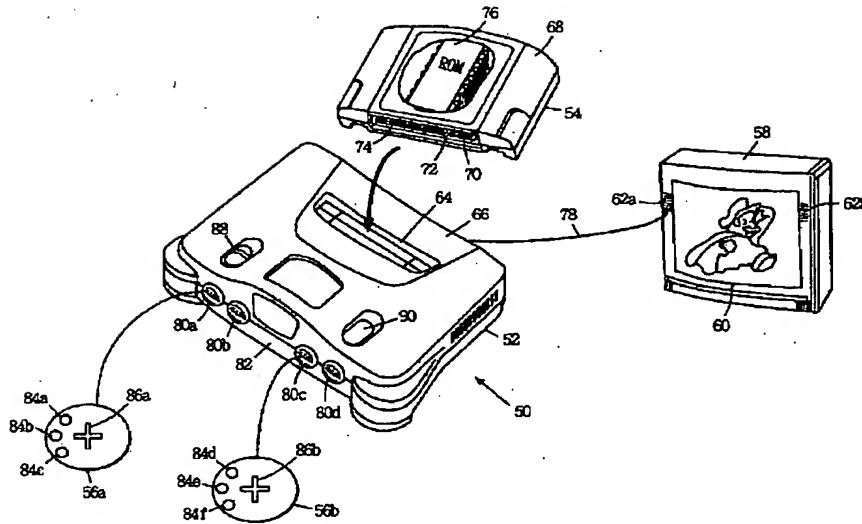
【符号の説明】

52…メインユニット
100…メインプロセッサ
200…コプロセッサ
300…メインメモリ
400…信号プロセッサ
500…表示プロセッサ

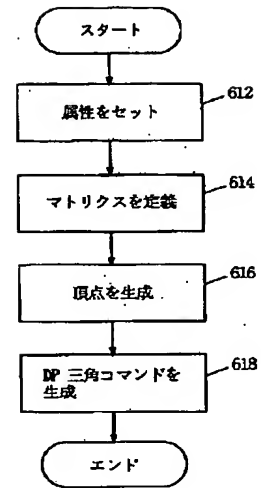
【图 45】



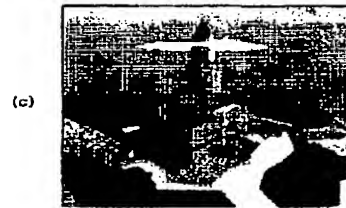
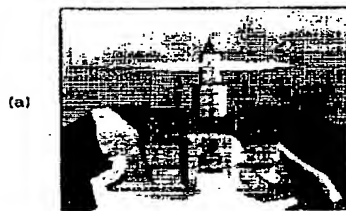
【図1】



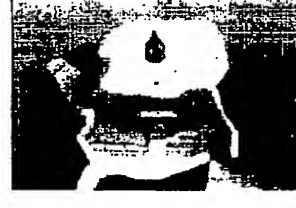
【図19】



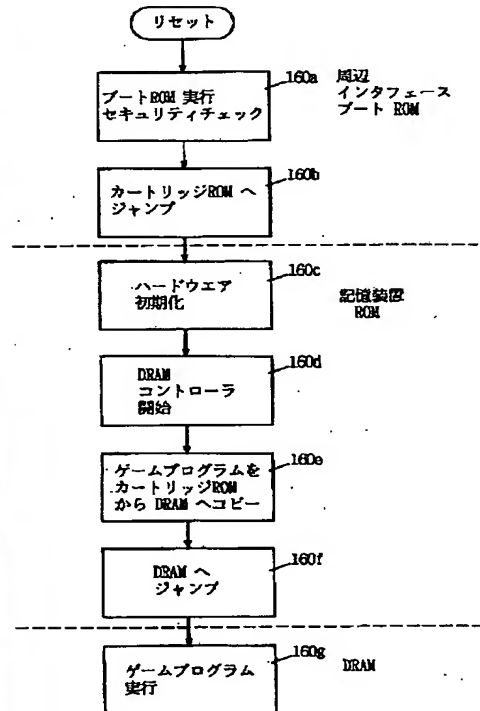
【図2】



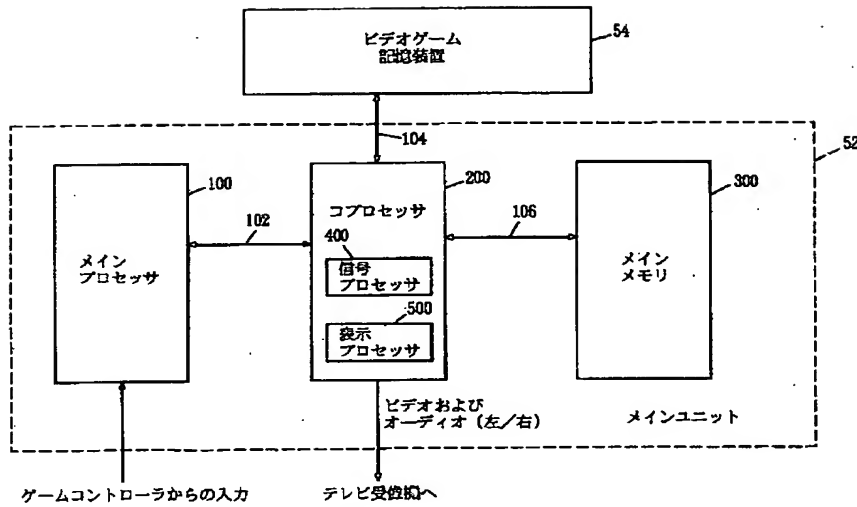
【図3】



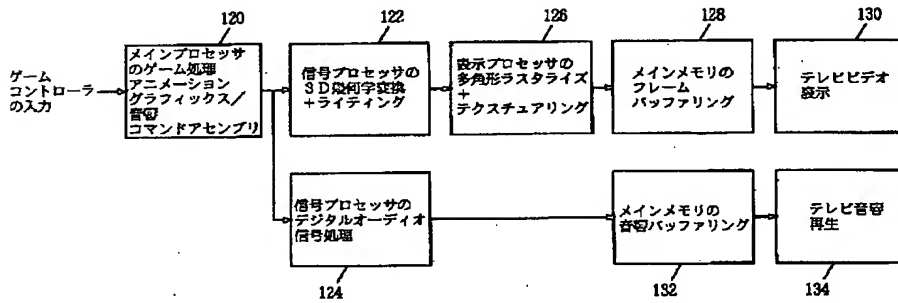
【図9】



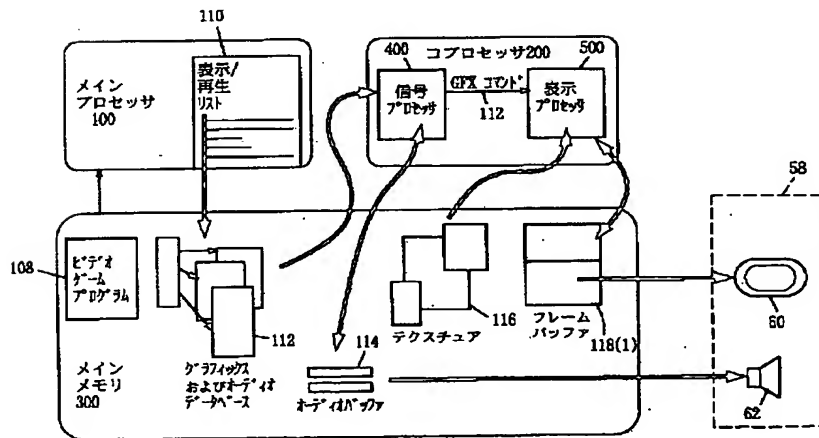
【図4】



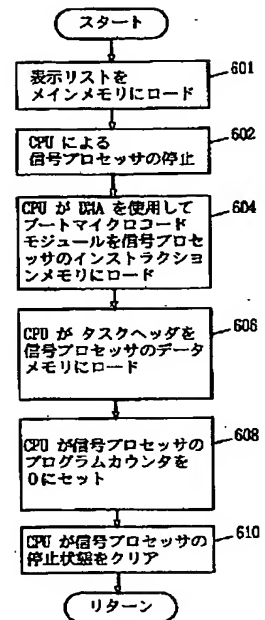
【図5】



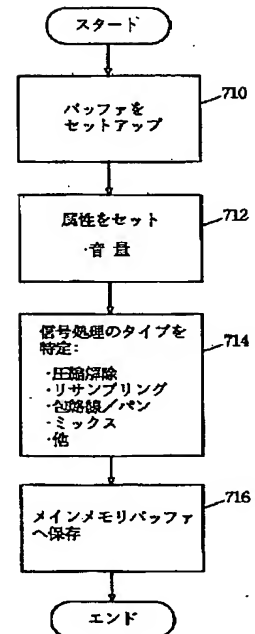
【図6】



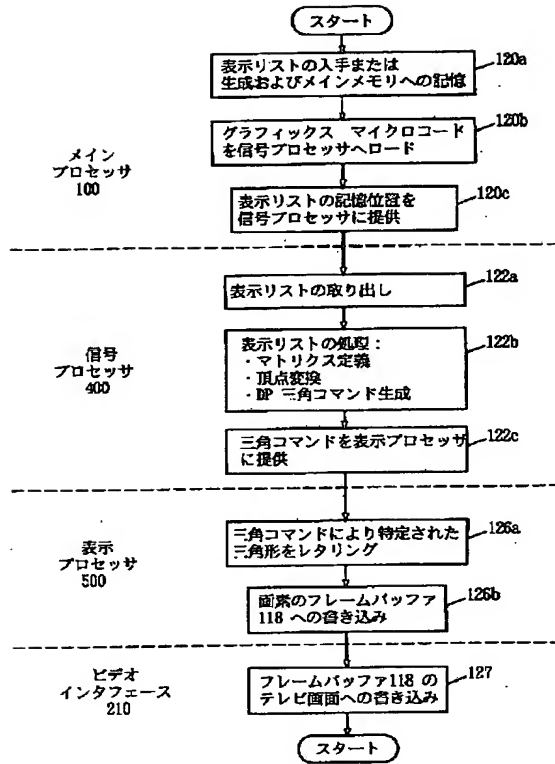
【図18】



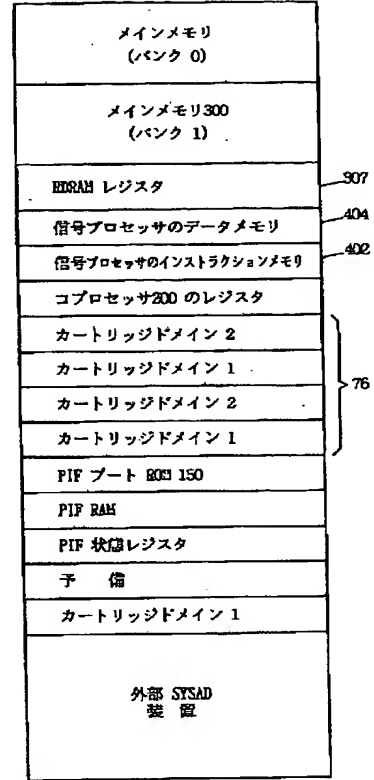
【図25】



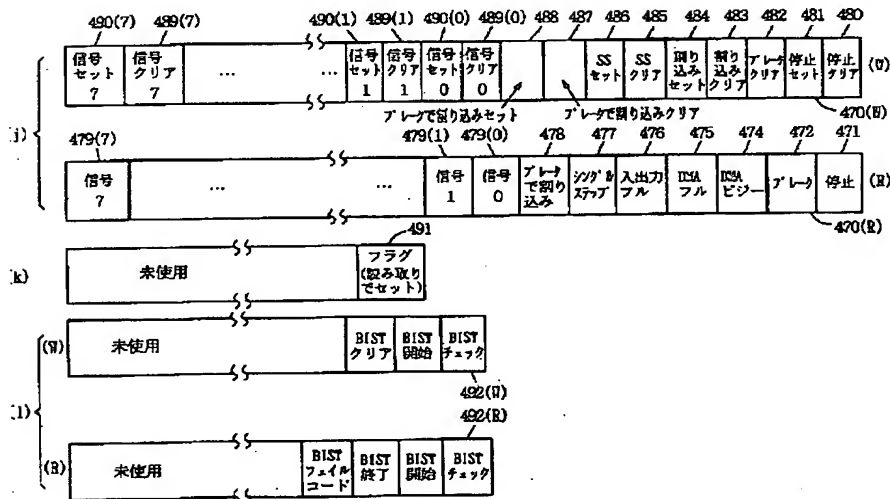
【図7】



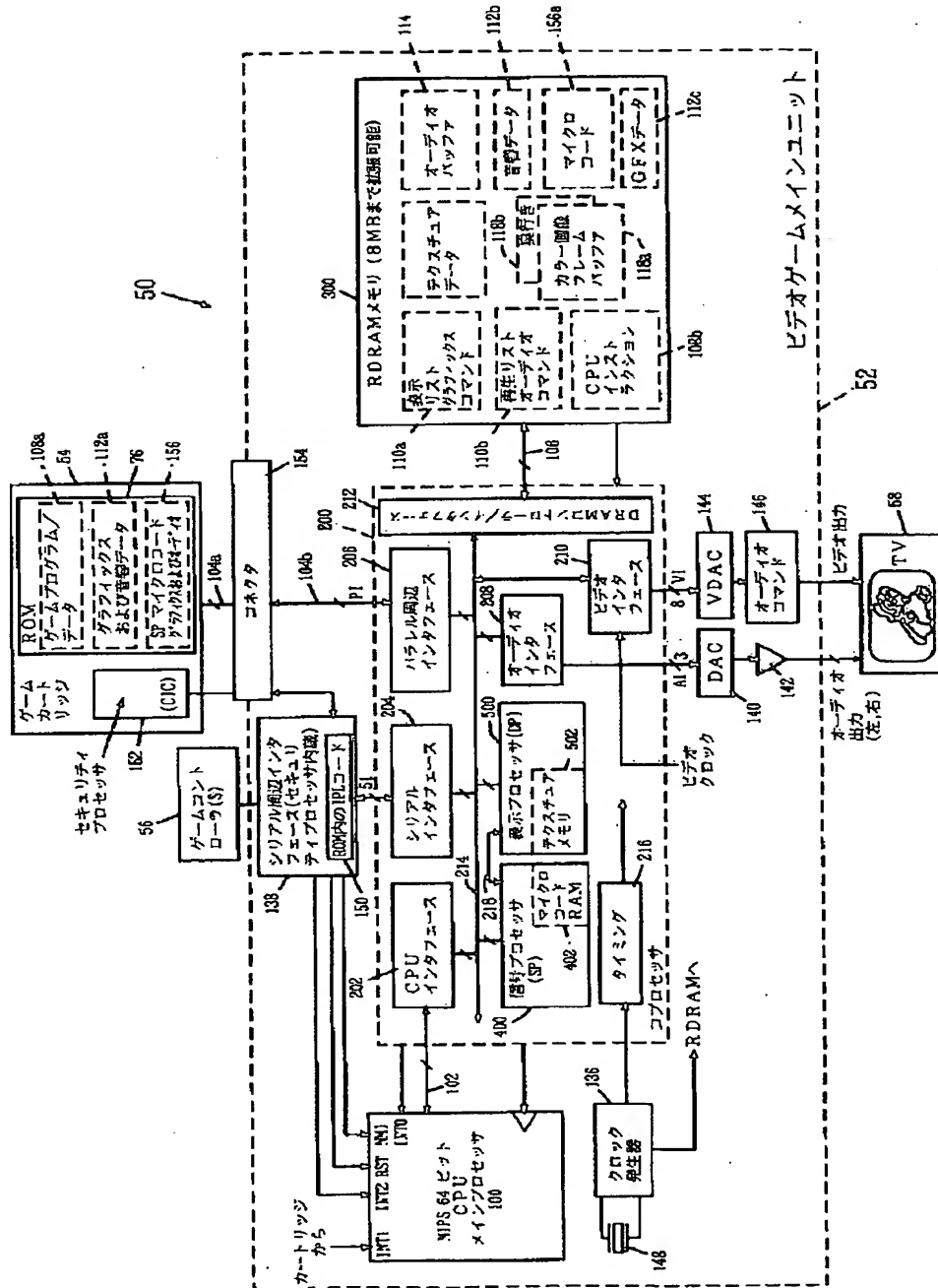
【図10】



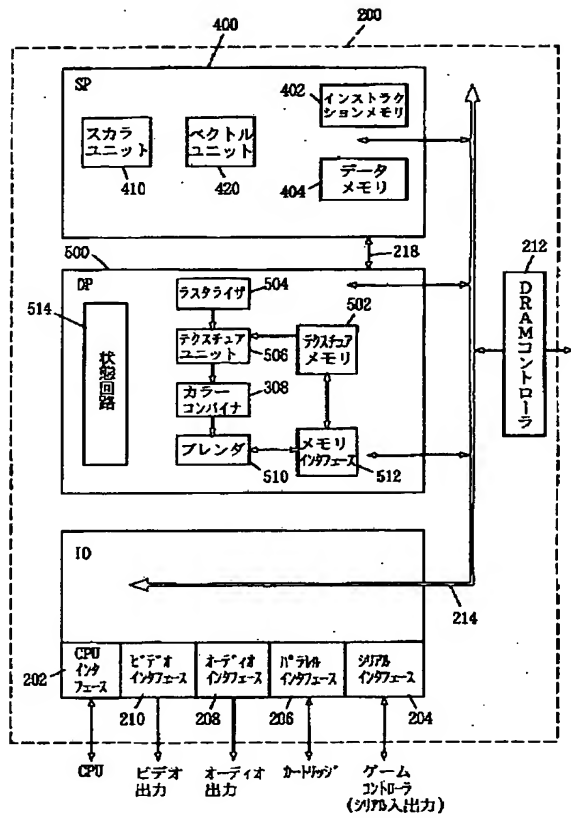
【図16】



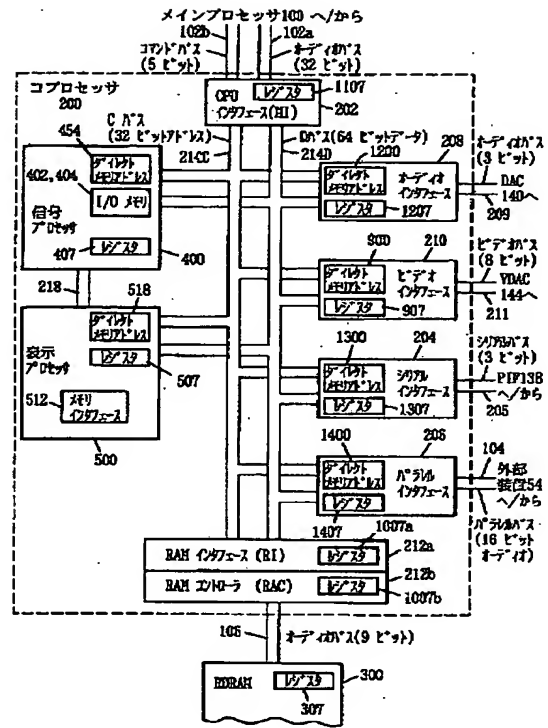
【図8】



【図11】

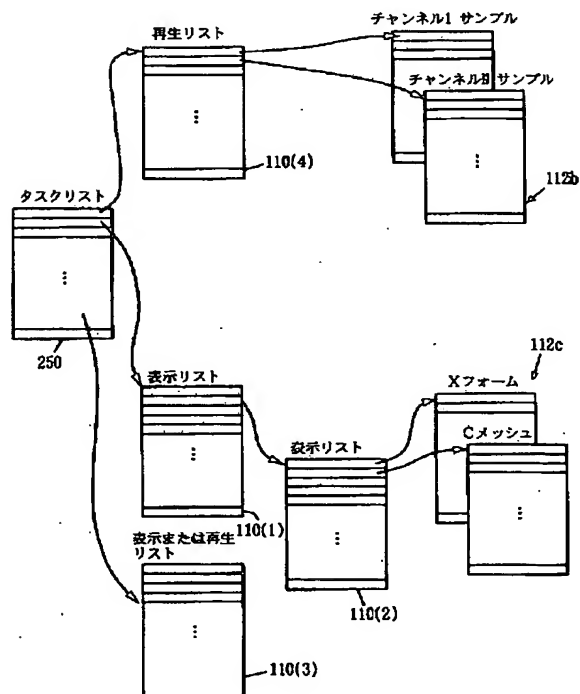
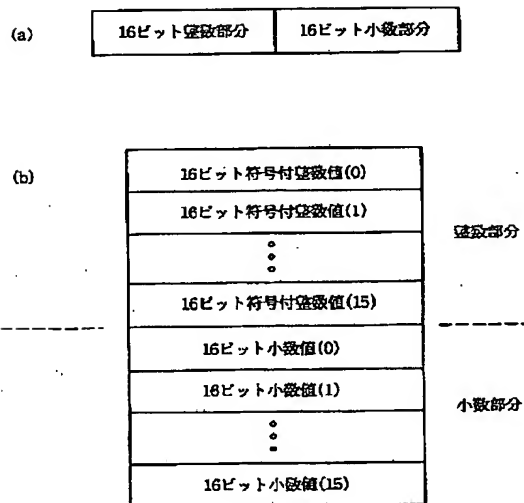


【図12】

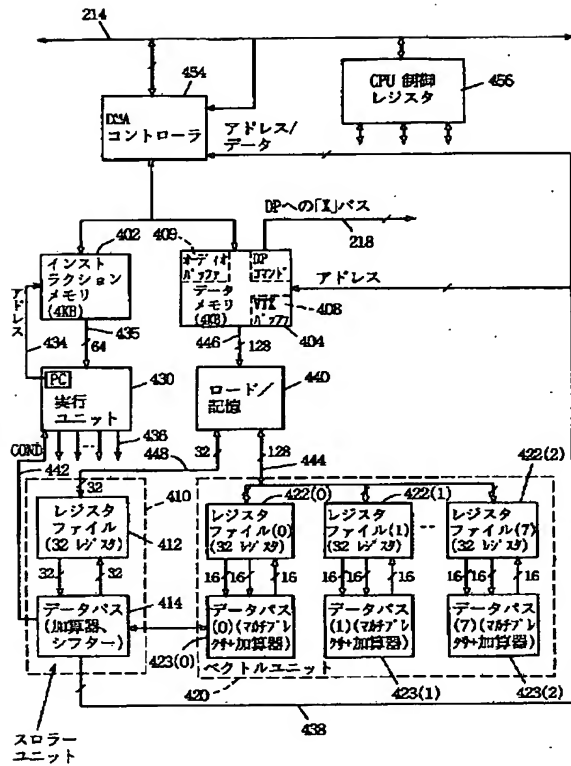


【図17】

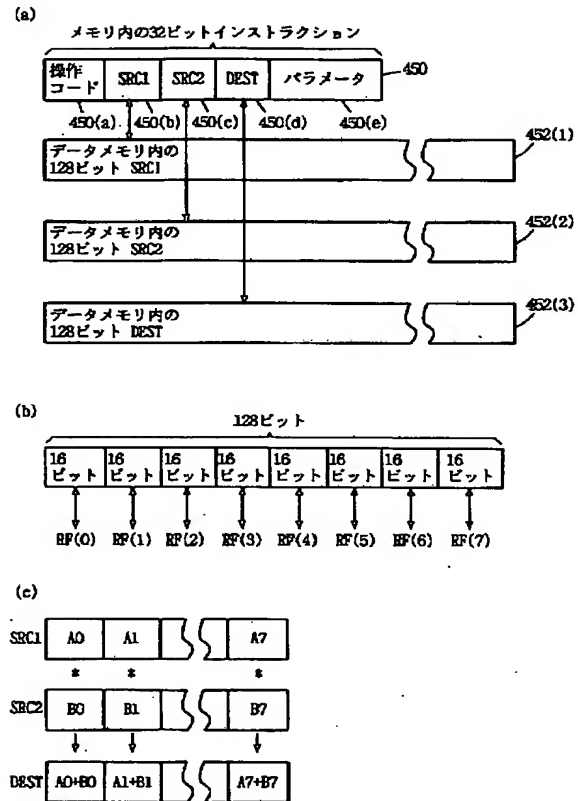
【図21】



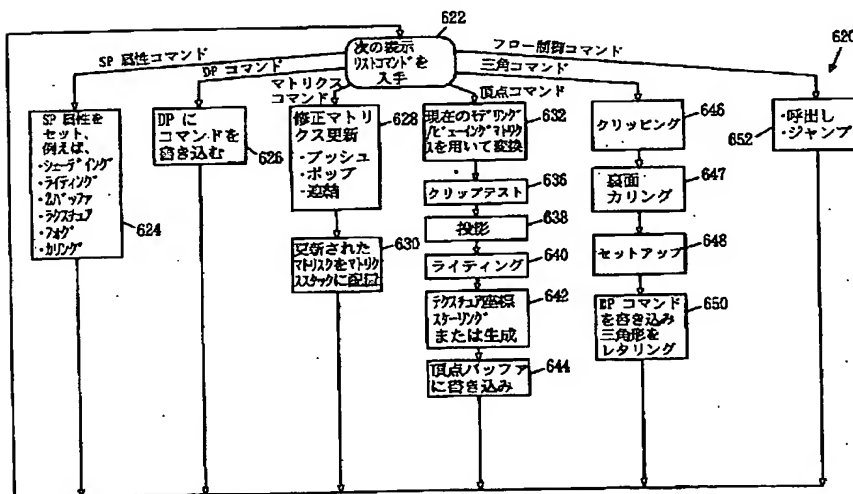
【図13】



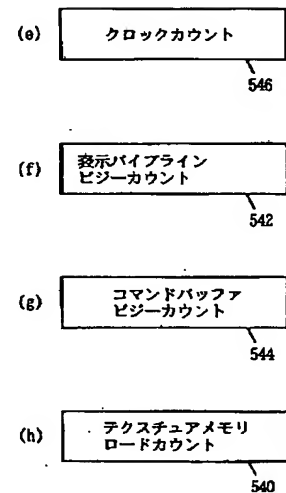
【図14】



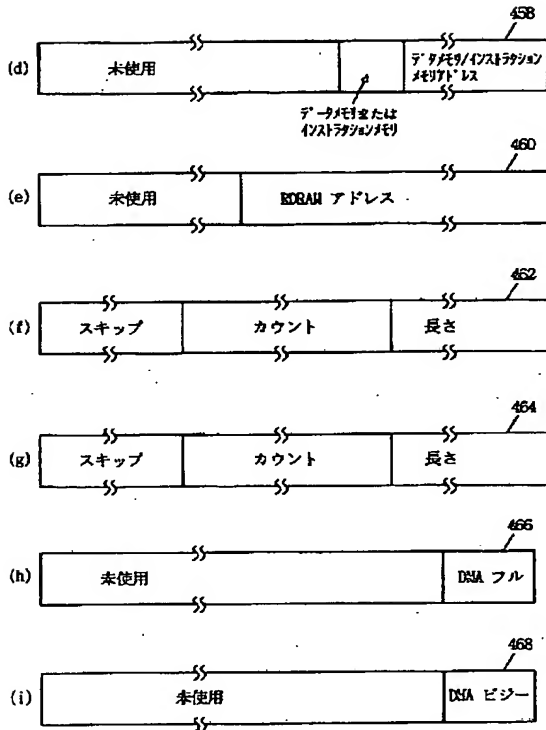
【図20】



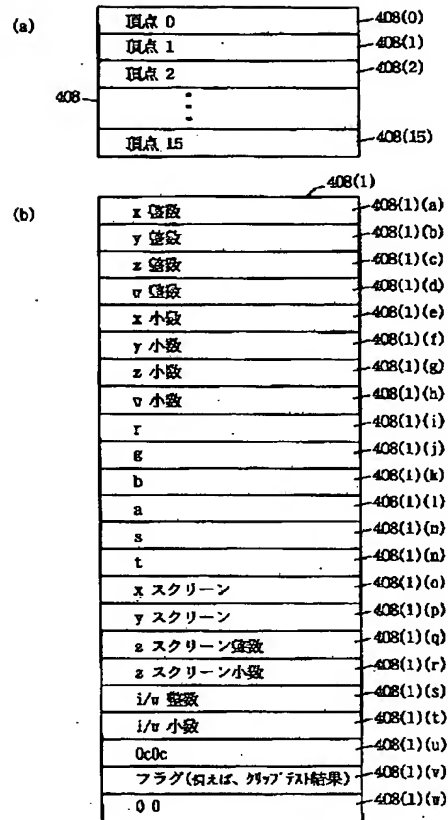
【図33】



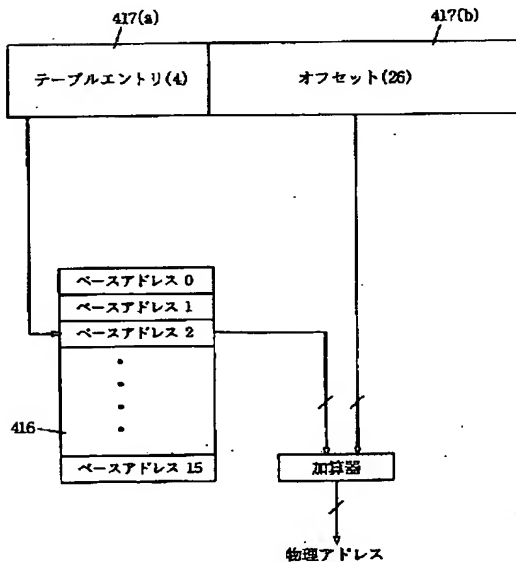
【図15】



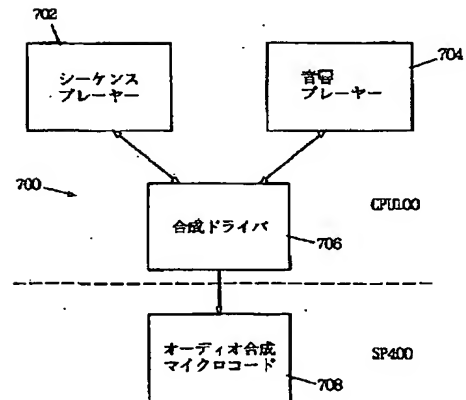
【図22】



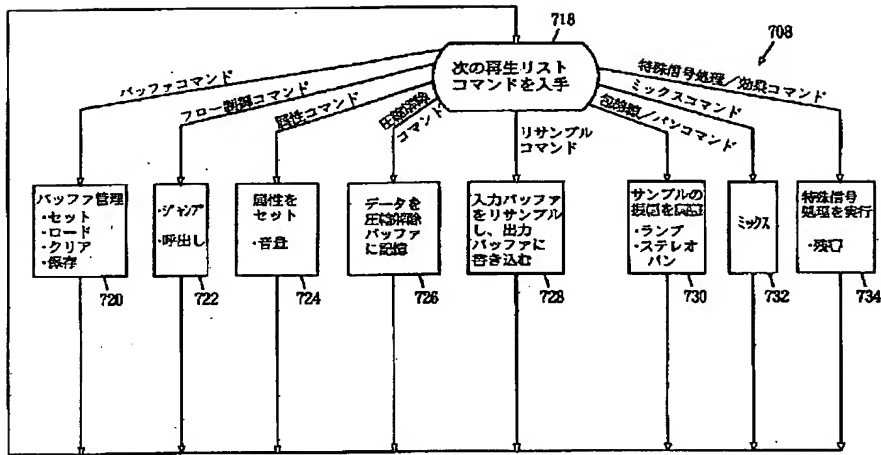
【図23】



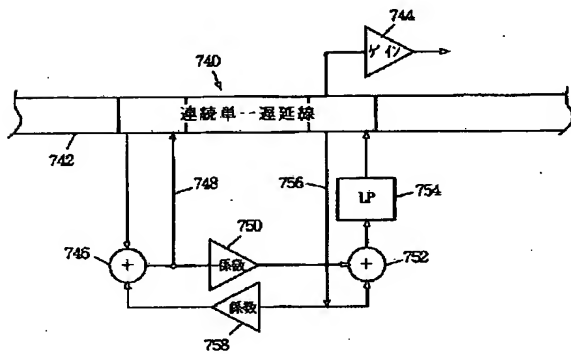
【図24】



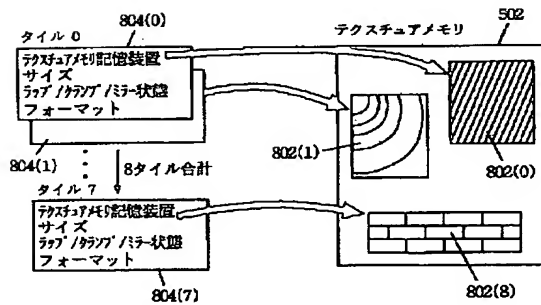
【図26】



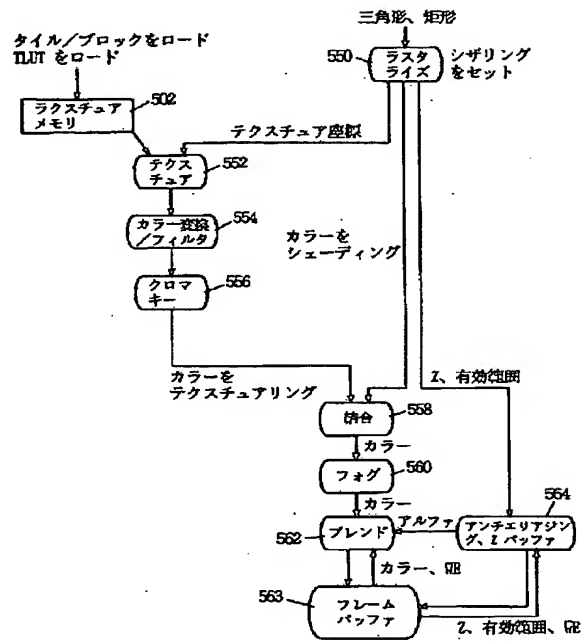
【図27】



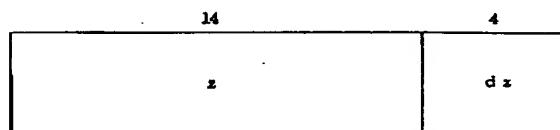
【図35】



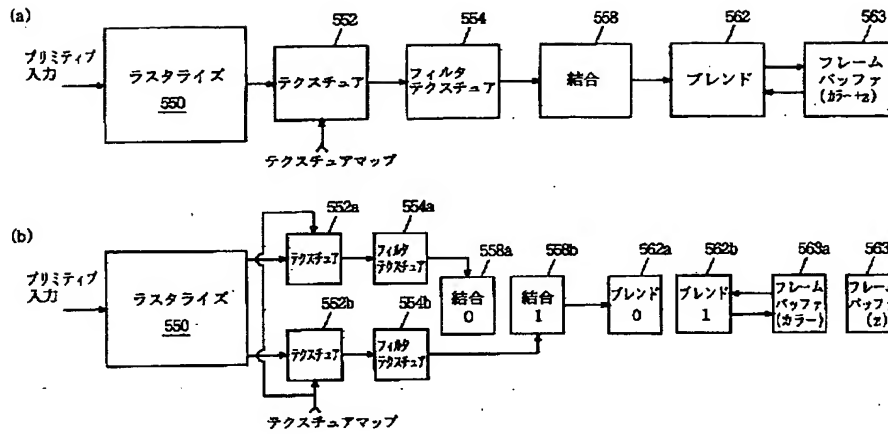
【図28】



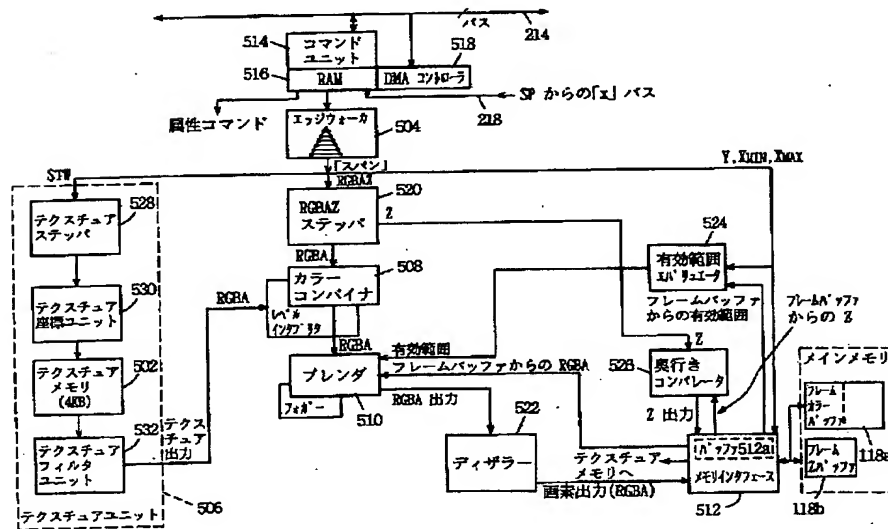
【図46】



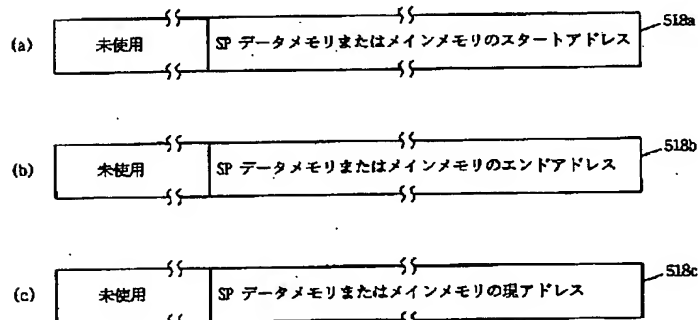
【図29】



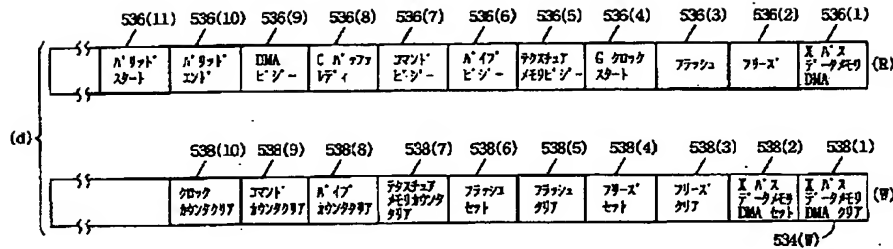
【図30】



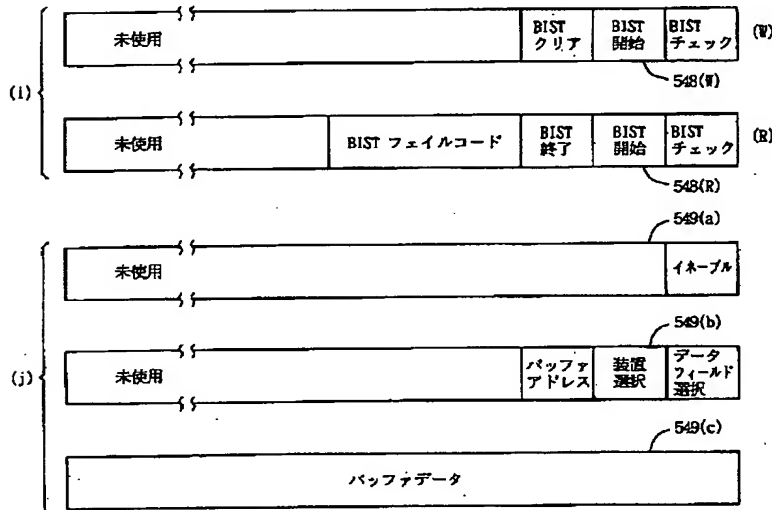
【図31】



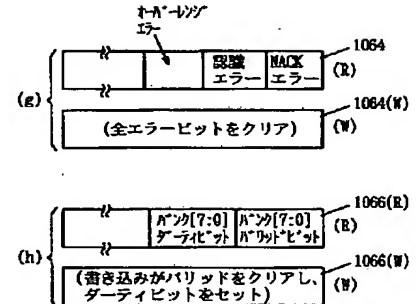
【図32】



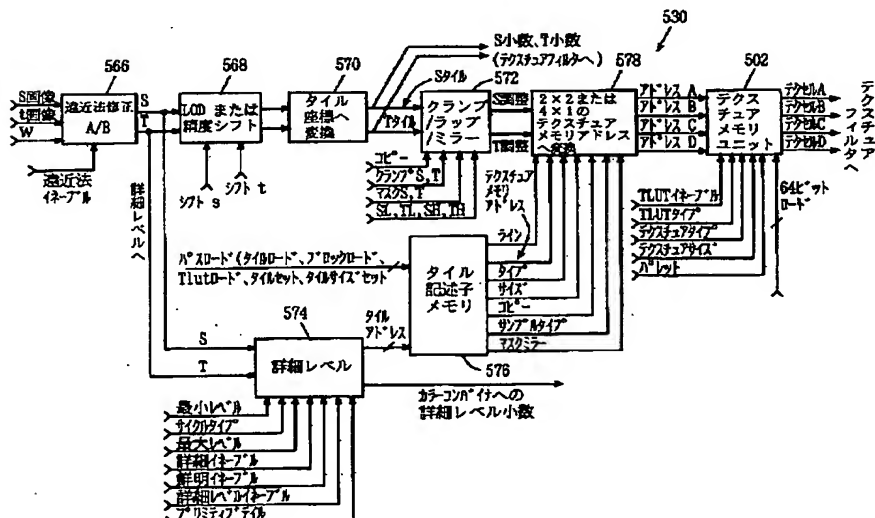
【図34】



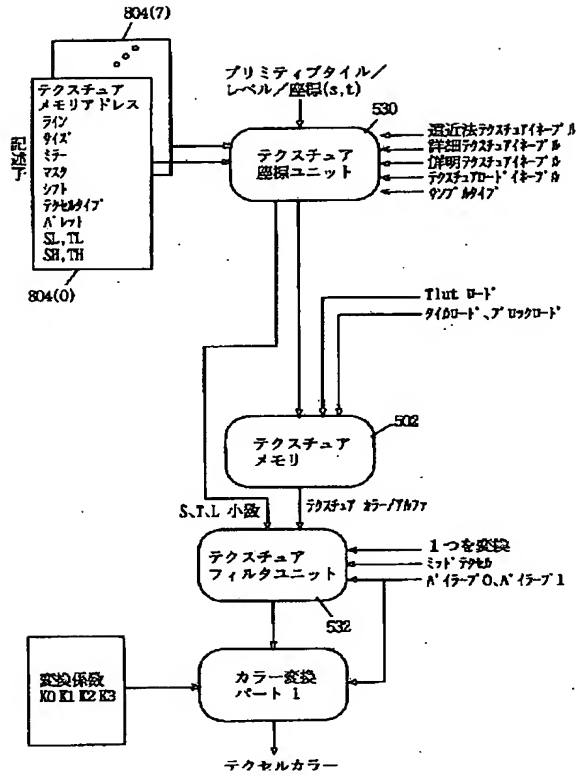
【図55】



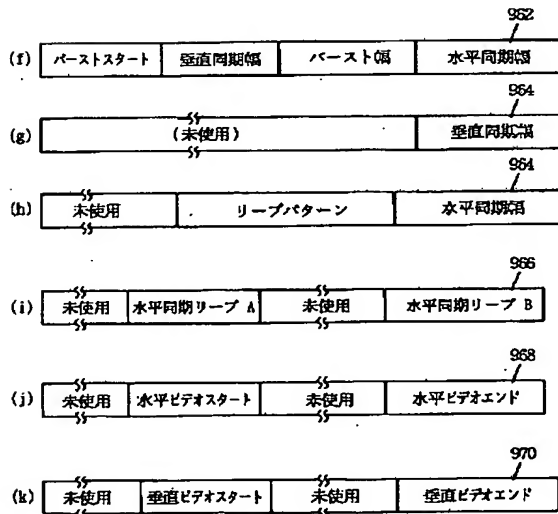
【図37】



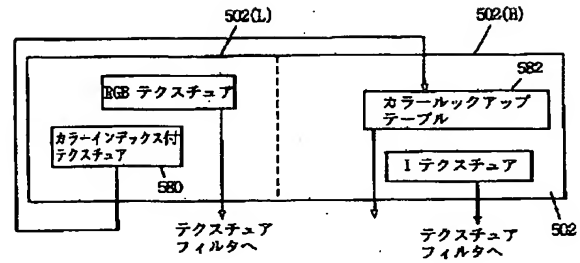
【図36】



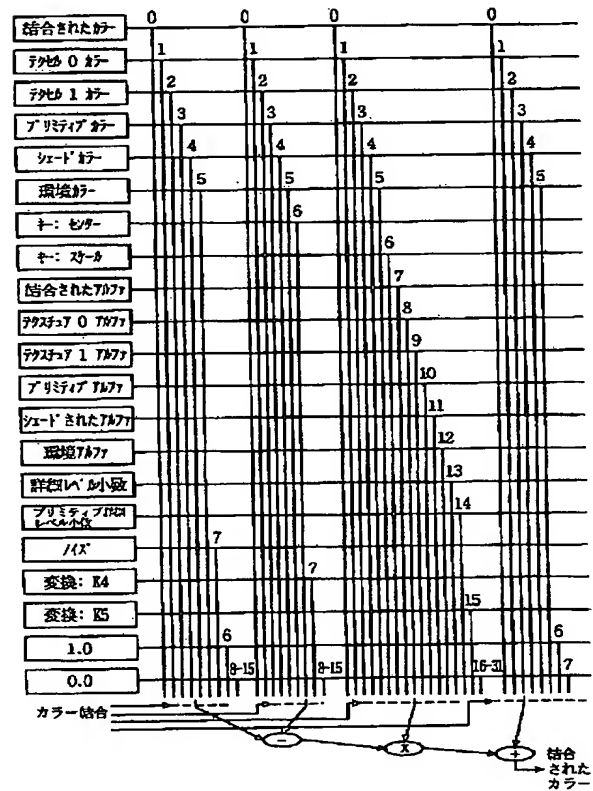
【図51】



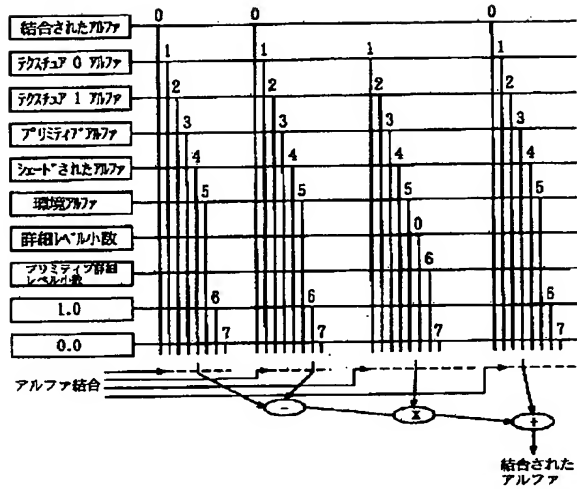
【図38】



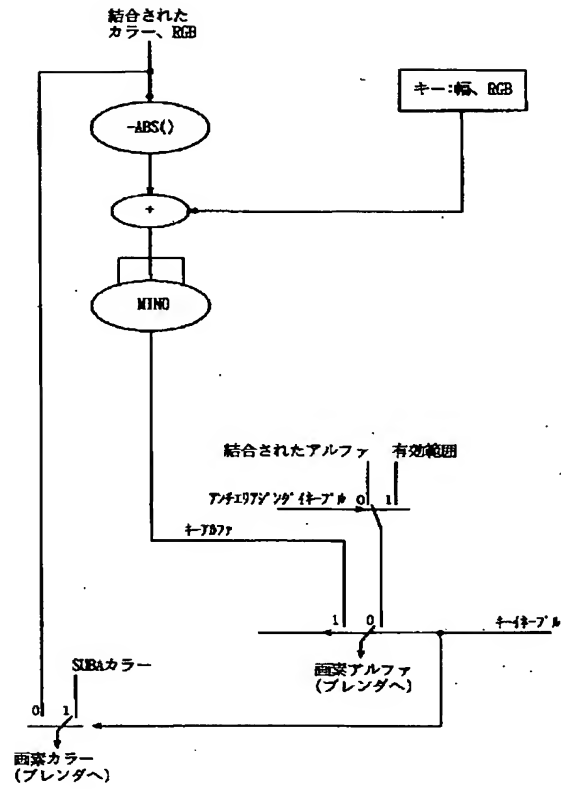
【図40】



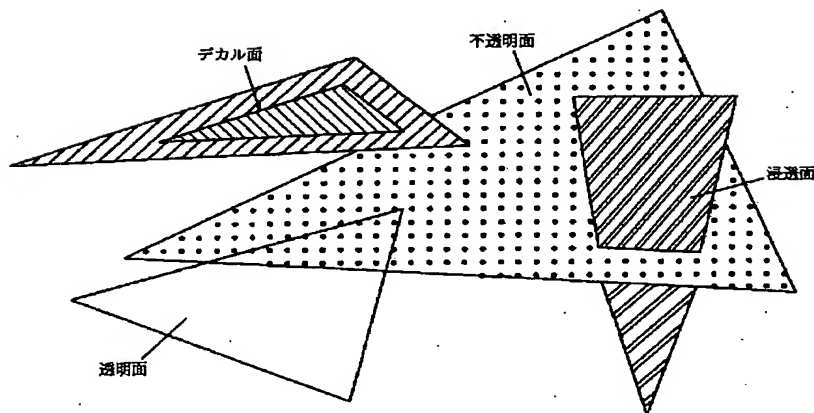
【図41】



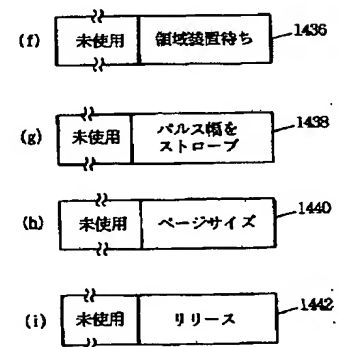
【図42】



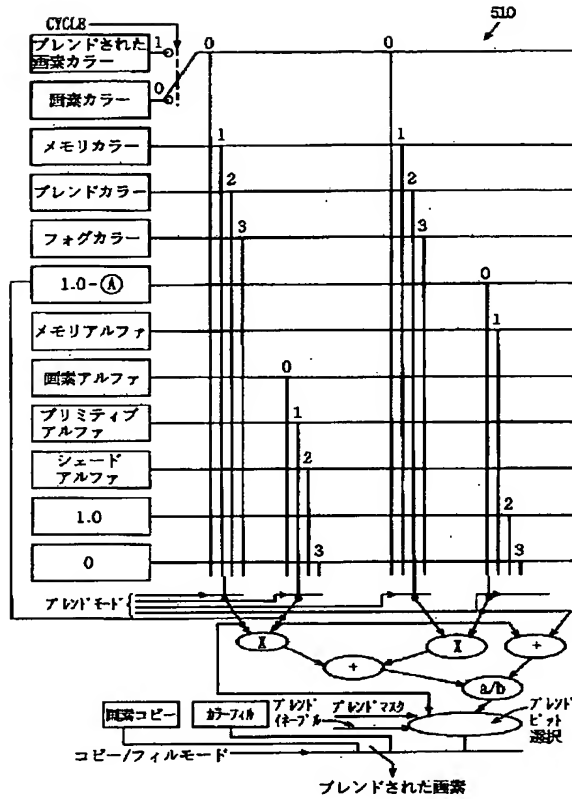
【図43】



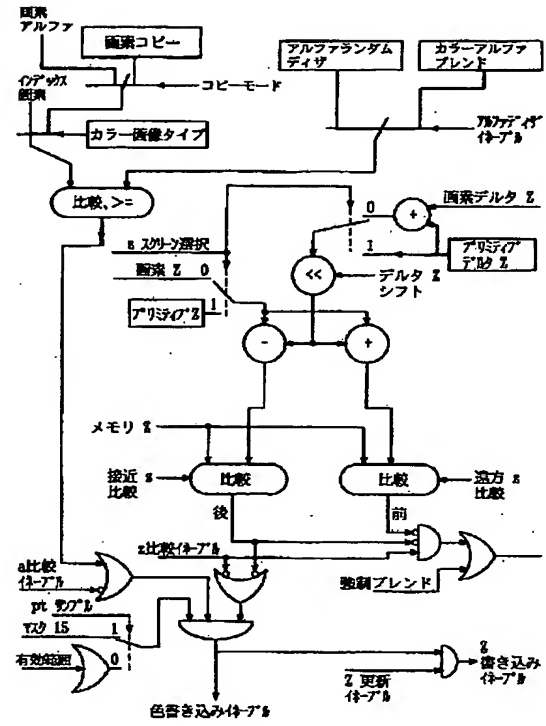
【図64】



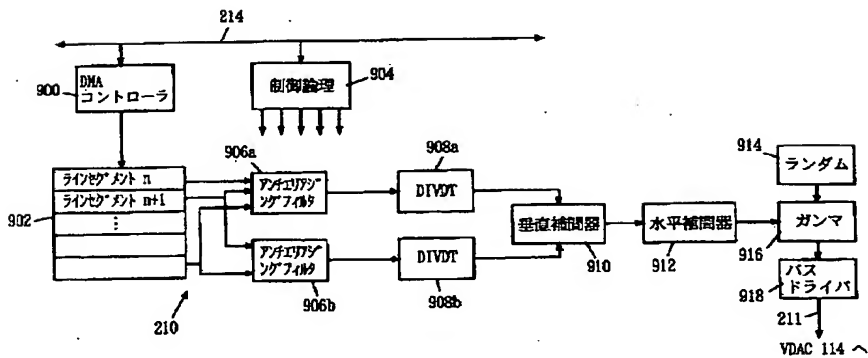
【圖 44】



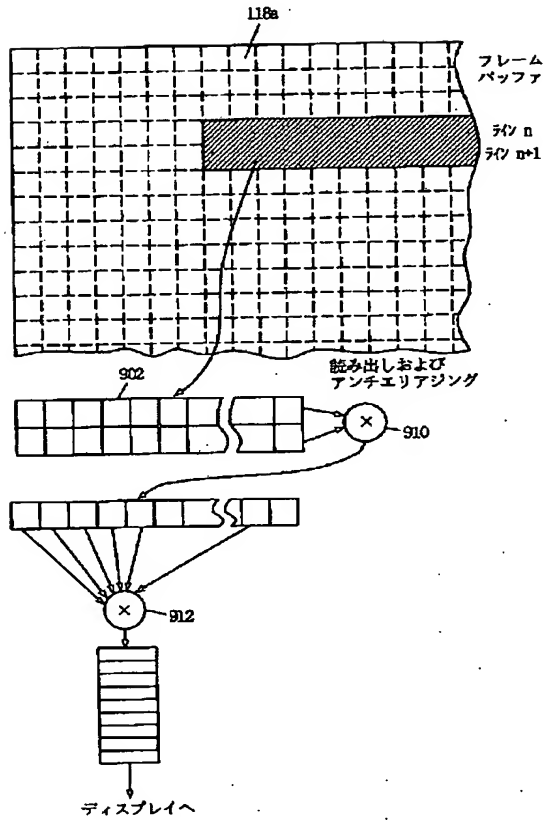
【圖 47】



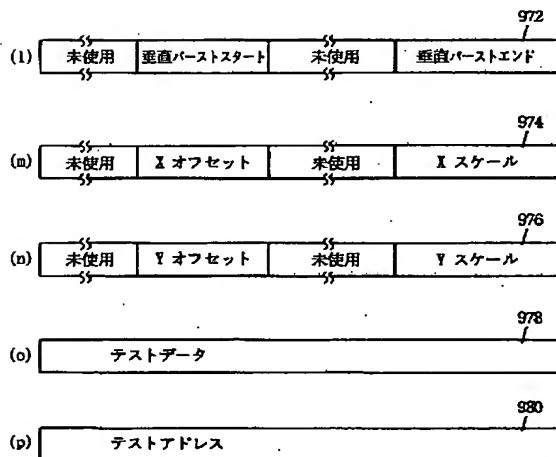
【圖 48】



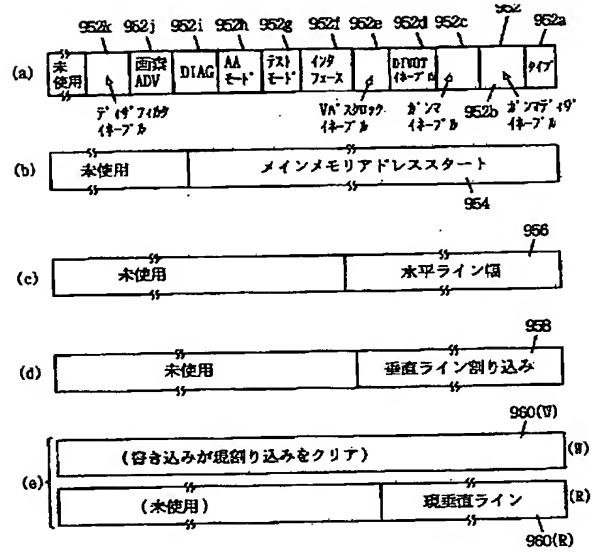
【図49】



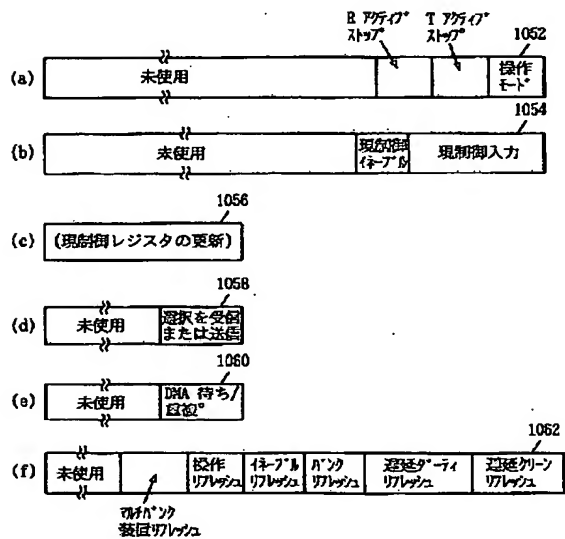
【図52】



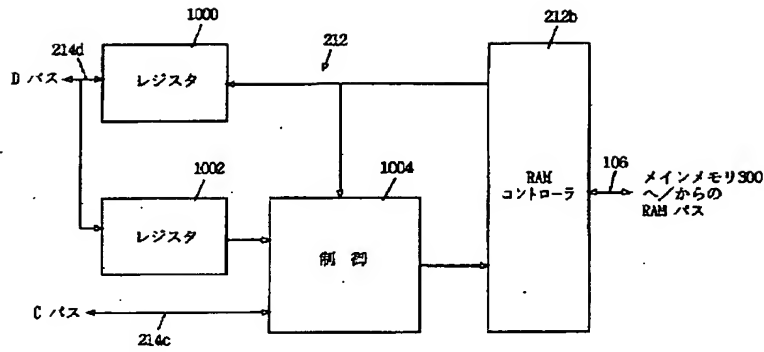
【図50】



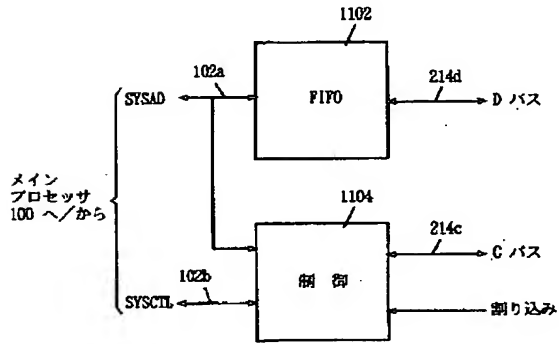
【図54】



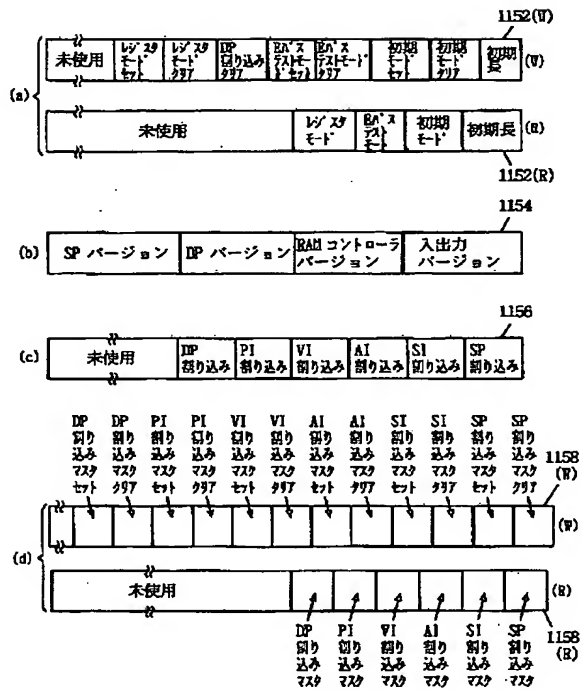
【圖 5 3】



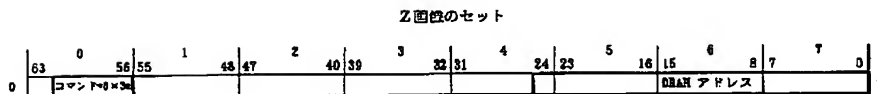
【圖 5 6】



【图 57】



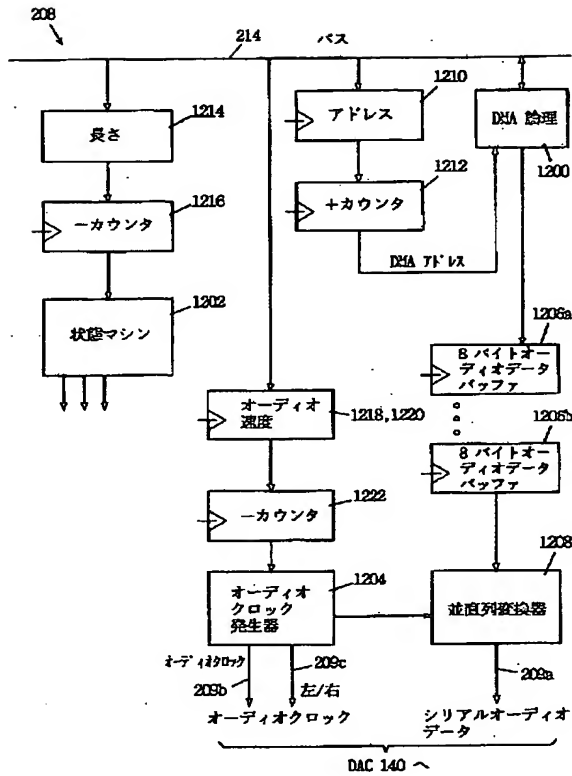
【圖 67】



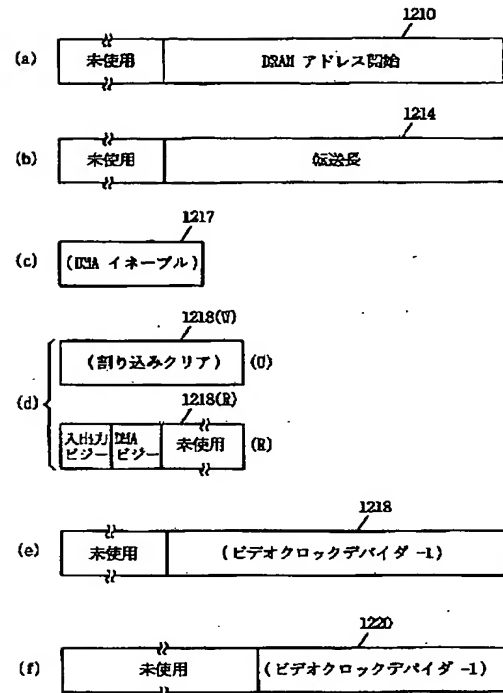
マスク画像のセットコマンドフォーマット

フィールド	ワード	ビット	説明
コマンド	0	61-58	コマンド識別子
DRAM アドレス	0	26-0	DRAM内の4096のベースアドレス (左上4096)、バイト単位

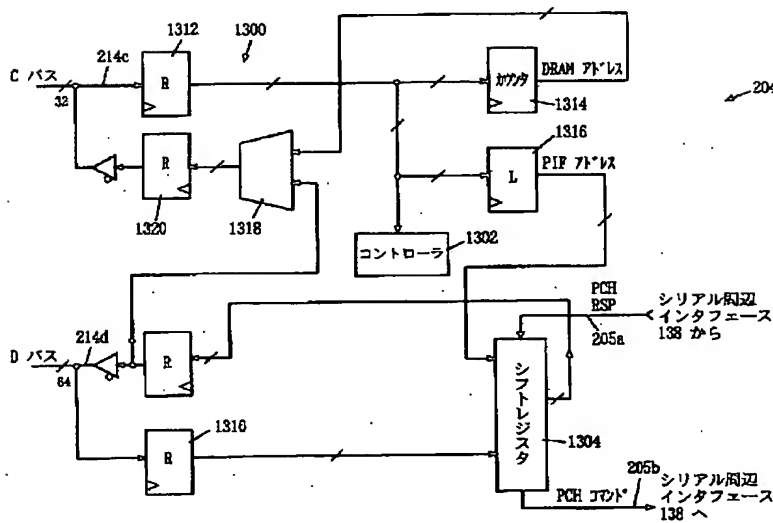
【図58】



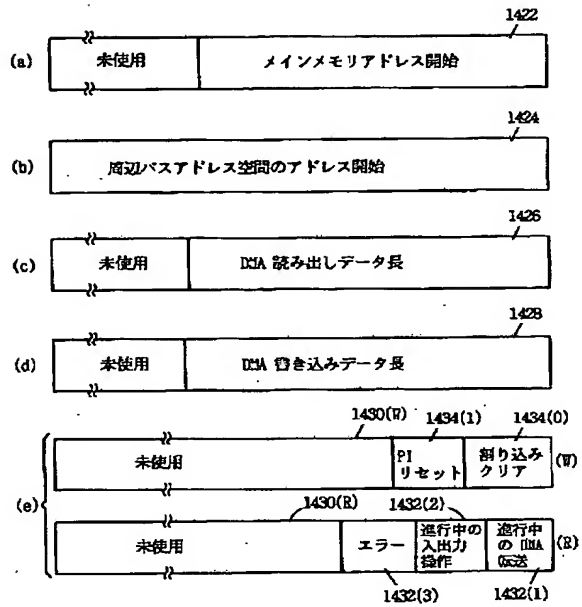
【図59】



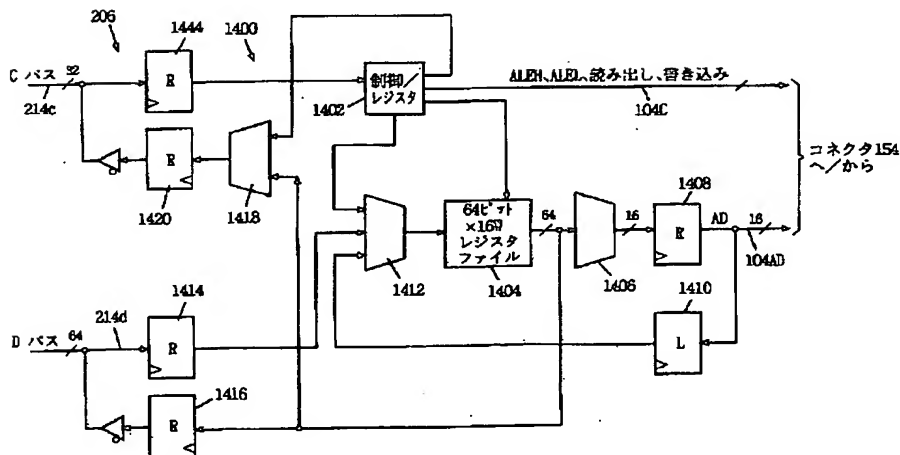
【図60】



【图63】



【圖62】



カラー画像のセット

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
コマンド=0x3f																						0
フォーマット サイズ																						0
幅																						0
DRAM アドレス																						0

カラー画像のセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子
フォーマット	0	55-53	画像データフォーマット、0=rgba、1=yuv、2=カラーインデックス、3=1A、4=1
サイズ	0	52-51	画像/テクセルカラー要素のサイズ、0=4ビット、1=8ビット、2=16ビット、3=32ビット
幅	0	41-32	画像内の画像幅、画像縦=幅+1
DRAM アドレス	0	25-0	DRAM内の画像のベースアドレス (左上方隅)、バイト単位

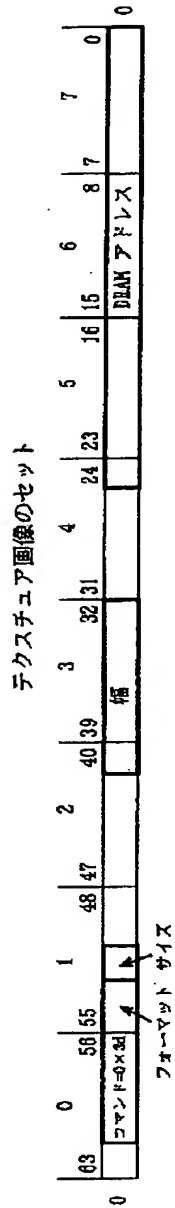
カラー画像のタイプ/サイズ

タイプ	8ビット	16ビット	32ビット
RGBA		✓	✓
YUV			
パインデックス	✓		

カラー画像のセット使用に関する注釈

実行きバッファアを用いての32ビットカラー画像の読み出し/修正/書き込みは、2サイクルモードで行わなければならない。

【図66】



テクスチャ画像のセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子
フォーマット	0	55-53	画像データフォーマット、0=rgb、1=yuv、2=カラーインデックス、3=1A、4=I
サイズ	0	52-51	画像/テクセルカラー幅のサイズ、0=4ビット、1=8ビット、2=16ビット、3=32ビット
幅	0	41-32	画像内の画像幅 画像幅=幅+1
DRAM アドレス	0	25-0	DRAM内の画像のベースアドレス (左上方隅)、バイト単位

テクスチャ画像のタイプ/サイズ

タイプ	4ビット	8ビット	16ビット	32ビット
RGBA			✓	✓
YUV			✓	
カラーインデックス	✓	✓		
1A	✓	✓	✓	
I	✓	✓	✓	

タイルのロープ

タイルのロードコマンドフォーマット

タイルのロード使用に関する注釈

4 ビットのカードスキューは、4 キロ以上のタイルをロードする際に、四位タイルスキューのセット「タイプ」フィールドを用いて、バイトとして（当該バイトは、一位上に配列されていないけれども）ロードされなければならない。すなわち、ブロックのロードパラメータが8 ビットのタイルスキューを有することを意味している。タイルのセットのタイプフィールドは、適正な4 ビットのタイプをセットするに利用していることが、タイルIDのセットは、ブロックのセットの位に2進正なSL, SR, LR, RRをセットするために用いることが、一般的には、タイルスキューは、テクニカル面談のカードおよびタイルのセットの「フォーマット」および「サイズ」フィールドを適正に記述することによって、ある1つのタイプとしてロードし、別のタイプとして応用することができる。通常、ロード中は、タイルのテクニカルサイズおよび面談の口腔スキューは合致しなければならない。

YUV 3L は偶数、3B は奇数でなければならない。つまり、Yは偶数であり、各テクセルにおいて評価を提供することを意味している。パイリニア化間を行うためにタイル3Bをタイルロード3B-1に引き合わせなければならない。それにより、0Y位においてクランプが発生する。

YVおよび32ビットZBA102に対して、ソフトウェアは、テクスチャメモリのローハーフにおいてタイルのベースアドレスを特定し、(8ビットのYにおいて2048ベース単位である)ローハーフに適合するタイルをロードしなければならない。

YUVおよび32ビットRGBA画像に関して、テクスチャメモリ内に、カラーインデックスのTLUTを含めることはできない。

上項および月項の小数部分座標は、通常図-でなければならぬ。マルチタイル操作におけるサブ面取座標のオフセットに有効。

タイルサイズのセット

タイルサイズのセットコマンドフォーマット

タイルサイズのセット使用に関する注釈:

YIVテクスチャについては、SLは偶数、S2は奇数でなければならない(つまり、偶は偶数のテクセルであることになる)。ハードウェアは、奇数のテクセルで有効なYIVを与えるためにS2-1、すなわち最後の偶数のテクセルをクランプする。

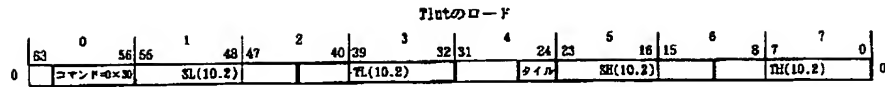
リニア插補が行われる命令、最大Sテクスチャ座標は奇数でなければならず、最大S+1において有効なUVを供給するために、SEは最大 S+2(奇数SE)である。

様々なタイプの三角コマンドは、下記の係数のグループを連結させることによって、構成される。連結の順序は、テーブルの左から右である。係数の各グループのフォーマットは、次の頁に示す。

三、命令

コマンド	エッジ	シェード	テクスチャ	Zバッファ
シェードなし三角形, 0x00	✓			
シェードあり三角形, 0x0e		✓		
テクスチャ済三角形, 0x0a	✓		✓	
シェードあり, テクスチャ済三角形, 0x0e	✓	✓		
シェードなし, Zバッファリング済三角形, 0x09	✓			✓
シェードあり, Zバッファリング済三角形, 0x0d	✓	✓		✓
テクスチャあり, Zバッファリング済三角形, 0x0b	✓	✓	✓	
シェードあり, テクスチャあり, Zバッファリング済三角形, 0x0f	✓	✓	✓	✓

【図72】



Tlutのロードコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-55	コマンド識別子。当該コマンドは、インデックス付テクスチャアルックアップテーブル(Tlut)の1128からのロードを開始するのに用いられる。当該テーブルは、テクスチャファイルタリング前にカラーインデックス付テクセルを参照しない。
SL	0	55-44	テーブル(0-255)へのローインデックス、小数部分のビットはゼロであるべき
TL	0	43-32	遷移ゼロ
タイル	0	28-24	タイル記述子インデックス
SH	0	23-12	テーブル(0-255)へのハイインデックス、小数部分のビットはゼロであるべき
TH	0	11-0	遷移ゼロ

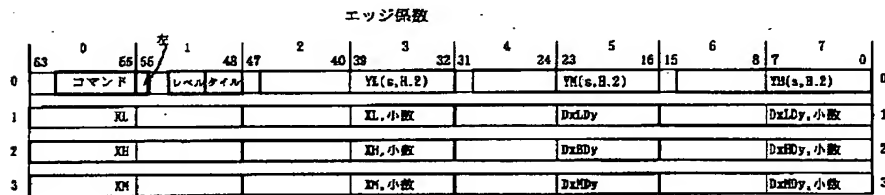
Tlutのロード使用に関する注釈:

DRAMアドレスを定義するには、16ビットのタイプと共に正確テクスチャのセットを使用すること。テクスチャメモリアドレスを定義するには、タイルのセットを使用すること。Tlutのロードのタイルを参照のこと。テクスチャ座標ユニットは、ロードする際にテーブルを4倍にして、4のD×S倍を持たなければならない。

テクスチャメモリアドレスは、テクスチャメモリのハイハーフ (msb=1)に存在しなければならない。

テクスチャメモリアドレスは、64ビットワード内である。

【図74】



エッジ係数コマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-55	コマンド識別子
左	0	55	左メジャーフラグ、1=左メジャー、0=右メジャー
レベル	0	53-51	ミップマップ数 -1
ファイル	0	50-48	タイル記述子インデックス。当該プリミティブのテクスチャを参照するために使用される。
YL	0	45-32	ローメインエッジのY座標
YH	0	29-16	ミッドメインエッジのY座標
YH	0	13-0	メジャーエッジのY座標
XL	1	63-48	ローエッジのX座標、整数
XL, 小数	1	47-32	ローエッジのX座標、小数
DxLdy	1	31-16	ローエッジの逆スロープ、整数
DxLdy, frac	1	15-0	ローエッジの逆スロープ、小数

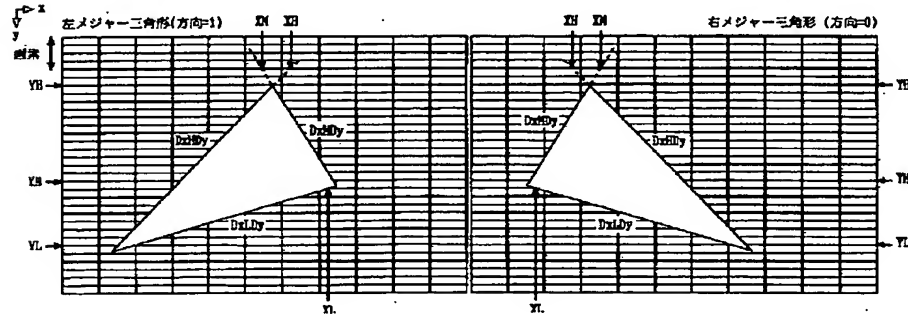
【図75】

エッジ係数コマンドフォーマット

フィールド	ワード	ビット	詳細
XH	2	63-48	メジャーエッジのX座標、整数
XH, 小数	2	47-32	メジャーエッジのX座標、小数
DxRdy	2	31-16	メジャーエッジの逆スロープ、整数
DxRdy, 小数	2	15-0	メジャーエッジの逆スロープ、小数
XH	3	63-48	ミドルエッジのX座標、整数
XH, 小数	3	47-32	ミドルエッジのX座標、小数
DxRdy	3	31-16	ミドルエッジの逆スロープ、整数
DxRdy, 小数	3	15-0	ミドルエッジの逆スロープ、小数

【図76】

エッジ係数は、ビュー画面上の特定のポイントにおいて計算される。下記の図は、各項が配置されている場所を示す。一般的に、Y項は、少なくともサブ画素(1/4画素)分解まで計算される。X項およびY項は、EおよびXエッジが前の走査ラインと交わる所で計算される。XLは、Lエッジが真中の頂点またはその下において、次のサブ画素と交わる所で計算される。



【図77】

シェード係数

	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	7	0
4		赤						緑						青						アルファ				
5		DrDx						DgDx						DbDx						DaDx				
6		赤 小数						緑 小数						青 小数						アルファ 小数				
7		DrDx, 小数						DgDx, 小数						DbDx, 小数						DaDx, 小数				
8		DrDe						DgDe						DbDe						DaDe				
9		DrDy						DgDy						DbDy						DaDy				
10		DrDe, 小数						DgDe, 小数						DbDe, 小数						DaDe, 小数				
11		DrDy, 小数						DgDy, 小数						DbDy, 小数						DaDy, 小数				

シェード係数コマンドフォーマット

フィールド	ワード	ビット	詳細
赤	4	63-48	赤カラー構成要素、係数
緑	4	47-32	緑カラー構成要素、係数
青	4	31-16	青カラー構成要素、係数
アルファ	4	15-0	アルファ カラー構成要素、係数
DrDx	6	63-48	X座標における変化する赤の変化、係数
DgDx	5	47-32	X座標における変化する緑の変化、係数

【図81】

Zバッファ係数

	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	7	0
20		Z						Z, 小数						DaDx						DaDx, 小数				
21		DaDe						DaDe, 小数						DaDy						DaDy, 小数				

Zバッファ係数コマンドフォーマット

フィールド	ワード	ビット	詳細
Z	20	63-48	逆実行、係数
Z, 小数	20	47-32	逆実行、小数
DaDx	20	31-16	X座標における変化するZの変化、係数
DaDx, 小数	20	15-0	X座標における変化するZの変化、小数
DaDe	21	63-48	メジャーエッジに従ったZの変化、係数
DaDe, 小数	21	47-32	メジャーエッジに従ったZの変化、小数
DaDy	21	31-16	Y座標における変化するZの変化、係数
DaDy, 小数	21	15-0	Y座標における変化するZの変化、小数

【図78】

シェード係数コマンドフォーマット

フィールド	ワード	ビット	詳細
Ddx	5	31-18	X座標における変化毎の赤の変化、整数
Ddy	5	15-0	X座標における変化毎のアルファの変化、整数
S, 小数	6	63-48	Sカラー構成要素、小数
T, 小数	6	47-32	Tカラー構成要素、小数
W, 小数	6	31-16	Wカラー構成要素、小数
Alpha, 小数	6	15-0	Alphaカラー構成要素、小数
Ddx, 小数	7	63-48	X座標における変化毎の赤の変化、小数
Ddy, 小数	7	47-32	X座標における変化毎の青の変化、小数
Ddz, 小数	7	31-16	X座標における変化毎の青の変化、小数
Ddx, 小数	7	15-0	X座標における変化毎のアルファの変化、小数
DxDe	8	63-48	エッジに従った赤の変化、整数
DyDe	8	47-32	エッジに従った青の変化、整数
DzDe	8	31-16	エッジに従った青の変化、整数
DxDe	8	15-0	エッジに従ったアルファの変化、整数
DxDe	9	63-48	Y座標における変化毎の赤の変化、整数
DyDe	9	47-32	Y座標における変化毎の青の変化、整数
DzDe	9	31-16	Y座標における変化毎の青の変化、整数
DxDe	9	15-0	Y座標における変化毎のアルファの変化、整数
DxDe	10	63-48	エッジに従った赤の変化、小数
DyDe	10	47-32	エッジに従った青の変化、小数
DzDe	10	31-16	エッジに従った青の変化、小数
DxDe	10	15-0	エッジに従ったアルファの変化、小数
DxDe	11	63-48	Y座標における変化毎の赤の変化、小数
DyDe	11	47-32	Y座標における変化毎の青の変化、小数
DzDe	11	31-16	Y座標における変化毎の青の変化、小数
DxDe	11	15-0	Y座標における変化毎のアルファの変化、小数

【図79】

テクスチャ係数

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
12	S						T															未使用
13	Ddx						Ddy															未使用
14	S 小数						T 小数															未使用
15	Ddx, 小数						Ddy, 小数															未使用
16	DxDe						DyDe															未使用
17	DxDe						DyDe															未使用
18	DxDe, 小数						DyDe, 小数															未使用
19	DxDe, 小数						DyDe, 小数															未使用

テクスチャ係数コマンドフォーマット

フィールド	ワード	ビット	詳細
S	12	63-48	S テクスチャ座標、整数
T	12	47-32	T テクスチャ座標、整数
W	12	31-16	正規化座標実行、整数
Ddx	13	63-48	X座標における変化毎のSの変化、整数
Ddy	13	47-32	Y座標における変化毎のTの変化、整数
Ddz	13	31-16	X座標における変化毎のWの変化、整数

【図92】

フィルカラーのセット

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0	コマンド識別子																					圧縮されたカラー

フィルカラーのセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子
圧縮されたカラー	0	31-0	圧縮されたカラー、例えば、カラー図像が16ビットRGBAにセットされている場合、フィルカラーは、水平方向で隣接する2つの16ビットRGBA図像である。

【図80】

テクスチャ係数コマンドフォーマット

フィールド	ワード	ビット	詳細
S, 小数	14	63-48	S テクスチャ座標, 小数
T, 小数	14	47-32	T テクスチャ座標, 小数
V, 小数	14	31-16	正規化座標行, 整数, 小数
DsDx, 小数	15	63-48	X 座標における変化毎のSの変化, 小数
DtDx, 小数	15	47-32	X 座標における変化毎のTの変化, 小数
DsDy, 小数	15	31-16	X 座標における変化毎のVの変化, 小数
DsDe	16	63-48	エッジに従ったSの変化, 整数
DtDe	16	47-32	エッジに従ったTの変化, 整数
DsDe	16	31-16	エッジに従ったVの変化, 整数
DsDy	17	63-48	Y 座標における変化毎のSの変化, 整数
DtDy	17	47-32	Y 座標における変化毎のTの変化, 整数
DsDy	17	31-16	Y 座標における変化毎のVの変化, 整数
DsDe, 小数	18	63-48	エッジに従ったSの変化, 小数
DtDe, 小数	18	47-32	エッジに従ったTの変化, 小数
DsDe, 小数	18	31-16	エッジに従ったVの変化, 小数
DsDy, 小数	19	63-48	Y 座標における変化毎のSの変化, 小数
DtDy, 小数	19	47-32	Y 座標における変化毎のTの変化, 小数
DsDy, 小数	19	31-16	Y 座標における変化毎のVの変化, 小数

【図82】

矩形のフィル

0	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	8	8	7	7	0	0
0	コマンド識別子				XL(10,2)				YL(10,2)				XH(10,2)				YH(10,2)								0

矩形のフィルコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子
XL	0	55-44	矩形の右下方隅のX 座標
YL	0	43-32	矩形の右下方隅のY 座標
XH	0	23-12	矩形の左上方隅のX 座標
YH	0	11-0	矩形の左上方隅のY 座標

【図84】

矩形フリップのテクスチャ

0	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	8	8	7	7	0	0
0	コマンド識別子				XL(10,2)				YL(10,2)				XH(10,2)				YH(10,2)								0
1	S(s,0.5)				T(s,0.5)				DsDx(s,5,10)				DtDy(s,5,10)												1

矩形フリップのテクスチャコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子, S/T および DsDx/DtDyのハードウェア交換を除いて、矩形のテクスチャと同様
XH	0	55-44	矩形の左上方隅のX 座標
YH	0	43-32	矩形の左上方隅のY 座標
タイル	0	28-24	タイル記述子インデックス
XL	0	23-12	矩形の右下方隅のX 座標
YL	0	11-0	矩形の右下方隅のY 座標
S	1	63-48	矩形の左上方隅のS テクスチャ座標
T	1	47-32	矩形の左上方隅のT テクスチャ座標
DsDx	1	31-16	X 座標における変化毎のSの変化
DtDy	1	15-0	Y 座標における変化毎のTの変化

【図83】

矩形のテクスチャ																						
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
コマンド=0x24													XL(10.2)		YL(10.2)		XH(10.2)				YH(10.2)	0
0	S(s,0.5)												T(s,0.5)		DeDx(s,5.10)							1

矩形のテクスチャコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子
XH	0	55-44	矩形の左上隅のX座標
YH	0	43-32	矩形の左上隅のY座標
タイル	0	28-24	タイル記述子インデックス
XL	0	23-12	矩形の右下隅のX座標
YL	0	11-0	矩形の右下隅のY座標
S	1	63-48	矩形の左上隅のSテクスチャ座標
T	1	47-32	矩形の左上隅のTテクスチャ座標
DeDx	1	31-16	X座標における変位毎の32変化
DeDy	1	15-0	Y座標における変位毎の32変化

矩形のテクスチャの使用に関する注釈:

1つの画像をコピーするには、サイクルタイプをセットして、他のモードのセットに「コピー」し、タイルのセットの「シフトS」とDeDxとの組み合わせをセットし、4テクセルずつ実行すること。テクスチャファイルタリング、カラーコンバイン、またはブレンド操作は、コピーされたテクセルについては利用不可能である。書き込みイネーブルは、コピーされた各テクセルのアルファチャンネルのしきい値比較を用いて生成してもよい。

矩形のテクスチャおよびコピーモード:

コピーモードでは、バップアまたはアンチエイリアシングはない。

4ビット、YUVおよび32ビットのRGBAテクスチャは、直接コピーすることはできない。

4ビットの画像は、コピー前に8ビットの画像に拡張される。

4ビットおよび8ビットの画像は、8ビットのカラー画像のみにコピーすることができる。

16ビットの画像は、16ビットのカラー画像のみにコピーすることができる。

4ビット、YUVおよび32ビットのRGBA画像は、8ビットまたは16ビットのカラー画像と、正しく規格化した画像座標および幅を用いることによりコピーすることができる。

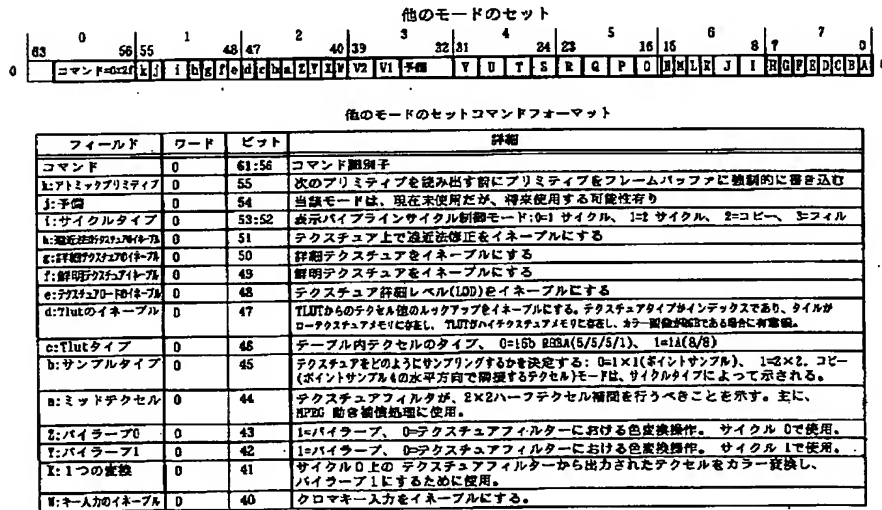
【図85】



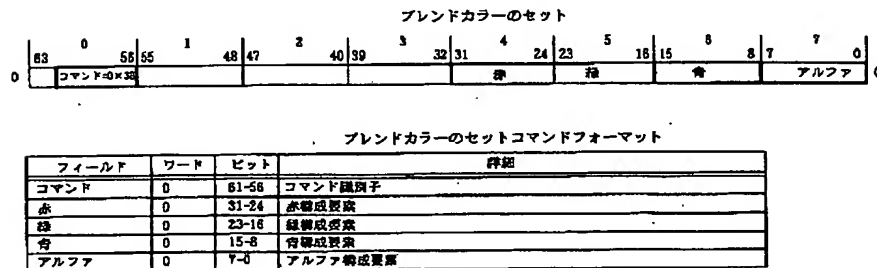
コンバインモードの使用に関する注釈:

カラーコンバインは、各カラーに(A-R)の式を適用する。RGBおよびアルファチャンネルは、列々のマルチプレクサ選択を有する。さらに、サイクル 0 および サイクル 1 には、列々のマルチプレクサ選択がある。RDPが1サイクルモードで構成されている場合、サイクル 0 および サイクル 1 のマルチプレクサ選択は同一の値にセットする。

【図86】



【図90】



【図87】

他のモードのセット

フィールド	ワード	ビット	詳細
V2:RGBディザ選択	0	39:38	0=マジックスクエアマトリックス(フィルタリングされている場合に好ましい) 1=「標準」ベイヤーマトリックス(フィルタリングされていない場合に好ましい) 2=ノイズ(依然と同様) 3=ディザなし
V1:アルファディザ選択	0	37:36	0=パターン 1=パターンのネガティブ 2=ノイズ 3=ディザなし
予備	0	35:32	将来使用するために確保されている。デフォルト値は 0x0。
V:b,m1a,0	0	31:30	ブレンドモードワード、1a入力選択を増大、サイクル 0
O:b,m1a,1	0	29:28	ブレンドモードワード、1a入力選択を増大、サイクル 1
T:b,m1b,0	0	27:26	ブレンドモードワード、1b入力選択を増大、サイクル 0
S:b,m1b,1	0	25:24	ブレンドモードワード、1b入力選択を増大、サイクル 1
R:b,m2a,0	0	23:22	ブレンドモードワード、2a入力選択を増大、サイクル 0
Q:b,m2a,1	0	21:20	ブレンドモードワード、2a入力選択を増大、サイクル 1
P:b,m2b,0	0	19:18	ブレンドモードワード、2b入力選択を増大、サイクル 0
O:b,m2b,1	0	17:16	ブレンドモードワード、2b入力選択を増大、サイクル 1
N:予備	0	15	当該モードは現在使用されていないが、将来使用する可能性有り。
M:強制ブレンド	0	14	強制ブレンドのイネーブル
L:アルファ有効範囲選択	0	13	画素アルファに対して有効範囲(または有効範囲*アルファ)を使用する。
K:有効範囲*アルファ	0	12	画素アルファおよび有効範囲に対して有効範囲*アルファを使用する。
J:Zモード [1:0]	0	11:10	0:不透明 1:透過、2:透明、3:デカル
I:有効範囲優先 [1:0]	0	8:9	0:クランプ(通常)、1:ラップ(フル有効範囲と見なされたもの)、 2:消去(有効範囲への作用)、3:保存(メモリ有効範囲を越え書きしない)。
B:有効範囲の色	0	7	有効範囲のオーバーフロー(透明画)の時のみカラーを更新
G:画素読出しのイネーブル	0	6	色/有効範囲読み出し/修正/書き込みメモリアクセスをイネーブルにする
F:Z更新のイネーブル	0	5	色書き込みがイネーブルにされている場合、Zの書き込みをイネーブルにする
E:Z比較のイネーブル	0	4	実行き比較における条件付き色書き込みのイネーブル
D:アチエリダングのイネーブル	0	3	強制ブレンドがされていなければ、ブレンドのイネーブルに有効範囲ビットを使用させる
C:ソースの選択	0	2	プリミティブまたは画素2を選択する
B:アルファ比較のイネーブル	0	1	アルファ比較においてランダムノイズを使用する、またはアルファ比較にブレンドアルファを使用する。
A:アルファ比較のイネーブル	0	0	アルファ比較における条件付き色書き込み。

他のモードのセットの使用に関する注釈:

YUV コピーモードはサポートされない。

「サイクルタイプ」のフィルモードは、フィルカラーをコピーすることを意味する(フィルカラーのセット参照)。

カラー画像が2バッファを指し示すようにセットし(カラー画像のセット)、最初の実行値で満たすことによって、2バッファの初期値を設定すること。

TLUTは全てのモードに影響を与えているので、TLUT内ではマルチタイル混合カラーインデックスおよび他のインデックスはサポートされない。

【図88】

環境カラーのセット

	0	1	2	3	4	5	6	7	8						
63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0
0	コマンド														アルファ

環境カラーのセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-55	コマンド識別子
0	0	31-24	画素形成要素
0	0	23-16	画素形成要素
0	0	15-8	画素形成要素
アルファ	0	7-0	アルファ形成要素

【図89】

プリミティブカラーのセット

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0	コマンド00h																					

プリミティブ最小レベル

プリミティブのセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	01-56	コマンド識別子
プリミティブ最小レベル	0	44-40	詳細または説明なテクスチャモード使用時の詳細レベル小段に対する最小クランプ、固定ポイント 0.5。
プリミティブレベル小段	0	39-32	プリミティブの詳細レベル小段、短形プリミティブに対するマルチタイル操作において主に使用される、0.8。
色	0	23-16	色組成要素
青	0	15-8	青組成要素
アルファ	0	7-0	アルファ組成要素

【図91】

フォグカラーのセット

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0	コマンド04h																					

フォグカラーのセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子
赤	0	31-24	赤組成要素
緑	0	23-16	緑組成要素
青	0	15-8	青組成要素
アルファ	0	7-0	アルファ組成要素

【図93】

プリミティブ実行のセット

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0	コマンド08h																					

プリミティブ実行のセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子
プリミティブ1	0	31-16	プリミティブ1
プリミティブデルタ1	0	15-0	プリミティブデルタ1

【図98】

フルシンク

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0	コマンド0Fh																					

フルシンクコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子。当該コマンドは、色値のdrawバッファが読み出されるか、または現在のプリミティブから書き込まれるまで、実行を停止させる。一般的には、表示バッファを切り換えたり、または、カラーレジスタを順番でテクスチャ図像として使用したりしてメモリデータを再使用する場合、もしくは、CPUからのdraw書き込み/読み出し図像への図像した読み出し/書き込みアクセスのためにメモリデータを再使用する場合にのみ、停止動作が必要となる。

【図94】

シザーのセット																								
	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0	
0	コマンド=0x2d				XB(10.2)				YB(10.2)				f0				XL(10.2)				YL(10.2)			

シザーのセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子
XB	0	55-44	画素空間内のシザーボックスの左上方隅のX座標
YB	0	43-32	画素空間内のシザーボックスの左上方隅のY座標
f	0	25	シザーフィールド、インターレースされた表示について奇数または偶数のラインのシザリングをイネーブにする。
0	0	24	奇数ライン:0=偶数ラインを保つ、1=奇数ラインを保つ、(インターレースされた表示に対して)全ての奇数ラインまたは全ての偶数ラインをスキップするべきか否かを示す。
XL	0	23-12	画素空間内のシザーボックスの右下方隅のX座標
YL	0	11-0	画素空間内のシザーボックスの右下方隅のY座標

【図95】

変換のセット																								
	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0	
0	コマンド=0x2e				K0(s1.7)				K1(s1.7)				K2(s1.7)				K3(s1.7)				K4(s1.7)			

変換のセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子。当該コマンドは、YUV 画素をRGBに変換するための係数を更新する。概念的には、式は以下の通り: $R = C0 * (Y - 16) + C1 * V$ $G = C2 * (Y - 16) + C3 * V$ $B = C4 * (Y - 16) + C5 * V$
K0	0	53-45	YUV-RGB変換マトリックスのK0項。
K1	0	44-36	YUV-RGB変換マトリックスのK1項。
K2	0	35-27	YUV-RGB変換マトリックスのK2項。
K3	0	26-18	YUV-RGB変換マトリックスのK3項。
K4	0	17-9	YUV-RGB変換マトリックスのK4項。
K5	0	8-0	YUV-RGB変換マトリックスのK5項。

変換のセット使用に関する注釈:

ハードウェアにおいて、色変換は2段階で行われる
テクスチャフィルタ(TF)では、以下の式が実行される:

$$R' = Y + K0 * V$$

$$G' = Y + K1 * V + K2 * V$$

$$B' = Y + K3 * V$$

カラーコンバイナにおいては、以下の式が実行される:

$$R = (R' - K4) * K5 * F$$

$$G = (G' - K4) * K5 * F$$

$$B = (B' - K4) * K5 * F$$

上記で定義されているK項における値は以下の通り:

$$K0 = C0 / D0$$

$$K1 = C2 / D0$$

$$K2 = C3 / D0$$

$$K3 = C4 / D0$$

$$K4 = (64 + 16 * ((C0 - 1.0) / 255))$$

$$K5 = 0.5$$

$$TFV = RGB変換の一般的な値は以下の通り:$$

$$K0 = 175$$

$$K1 = -43$$

$$K2 = -89$$

$$K3 = 222$$

$$K4 = 114$$

$$K5 = 42$$

【図96】

キー Rのセット																								
	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0	
0	コマンドのメモ												キー R(s7.4)				センター R				スケール R			

キーRのセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子。当該コマンドは、キー入力に用いる係数をセットする。キー入力に用いる式は次の通り: $キーR = クランプ(0.0, -abs((1 - センター) * スケール) * 幅, 1.0)$ キーアルファはキーR、キーG、キーBのうちの最小のものである。
K0 R	0	27-16	(ソフトエッジを含むキーウィンドウの半分のサイズ) * スケール。 幅 > 1.0の場合、当該チャンネルについてのキー入力は不可能になる。
センター R	0	15-8	キーがアクティブである場所において、カラーまたは強度を定義する。0-255
スケール R	0	7-0	(1.0 / (ソフトエッジのサイズ))。ハードエッジ入力についてはスケールを255にセットする。

【図97】

キーBのセット																							
63	0	58	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	7	0
0	コマンドのメモ			幅 G				幅 B			センター G			スケール G			センター B			スケール B			

キーBのセットコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-58	コマンド識別子。当該コマンドは、幅/高 キー入力に用いる係数をセットする。 概念的には、キー入力に用いる式は以下の通り： キーB/幅クランプ(0.0, -abs(6/8-センター)*スケール)+幅, 1.0)。 キーアルファは、キーB、キーG、キーRのうち最小のものである。
幅 G	0	55-44	(ソフトエッジを含むキーウィンドウの半分のサイズ)*スケール。 幅>1.0の場合、当該チャンネルについてのキー入力は、不可能になる。
幅 B	0	43-32	(ソフトエッジを含むキーウィンドウの半分のサイズ)*スケール。 幅>1.0の場合、当該チャンネルについてのキー入力は、不可能になる。
センター G	0	31-24	キーがアクティブである場所において、カラーまたは濃度を定義する、0-255。
スケール G	0	23-16	(1.0/(ソフトエッジのサイズ))。ハードエッジキー入力についてはスケールを255にセットする。
センター B	0	15-8	キーがアクティブである場所において、カラーまたは濃度を定義する、0-255。
スケール B	0	7-0	(1.0/(ソフトエッジのサイズ))。ハードエッジキー入力についてはスケールを255にセットする。

キーのセット使用に関する注釈:

ハードウェアにおいて、キー入力方程式は2段階で行われる。
カラーコンバイナ(OC)では、実行される式は以下の通り。

キー'=(幅高-センター)*スケール+0

アルファ手配ユニット(AT)において実行される式は以下の通り。

キー=クランプ(0, -abs(キー')*幅, 1.0)

キーアルファ=最小(キーB, キーG, キーR)

2サイクルモードでは、キー入力操作は、第2のサイクルにおいて特定されなければならない。(すなわち、キーアルファは、結合されたオペランドとしては利用できない。)

【図99】

ロードシンク																							
	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0	コマンドのメモ																						

フルシンクコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子。当該コマンドは、先のプリミティブが完全に終了するまでロードコマンド (TLUTのロード、タイルのロード、ブロックのロード)の実行を停止させる。遅延、ロードの時間によって、全てのロードコマンドを開始させる。

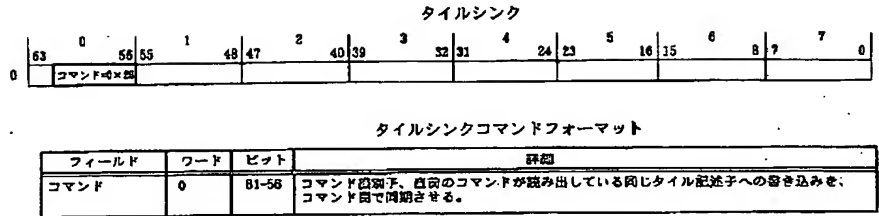
【図100】

パイプシンク																							
	63	0	58	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0	コマンドのメモ																						

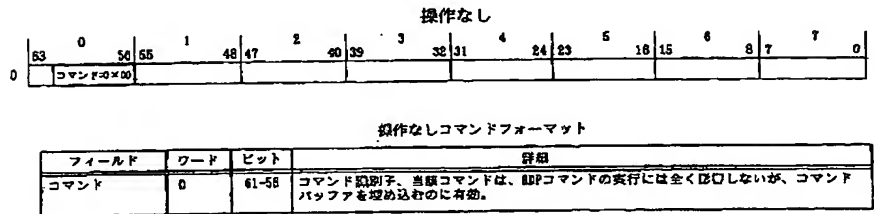
パイプシンクコマンドフォーマット

フィールド	ワード	ビット	詳細
コマンド	0	61-56	コマンド識別子。最高2つまでの先行するプリミティブによって読み出されている(プリミティブカラー/実行より)一般的な属性が、パイプシンクの後に続く。パイプシンクは、直前のプリミティブが過ぎるまで、いかなる属性の最終的な使用も停止させる。どのような数の属性コマンドの値にも、1つのパイプシンクが必要である。ソフトウェアは、現在何が読み出されているかがわかっている場合、パイプシンクの使用を最適化することができる。例えば、三角形または矩形がテクスチャ画像の属性を使用しないため、テクスチャ画像のセットは、パイプシンクがなくても、タイルのロードのために前処理の三角形または矩形に続くことができる。一般的に、RSPは、同じ属性をセットするとともに、提供された多数のプリミティブに対して最適化される。属性がプリミティブ毎に変化する場合は、性能は多少低下する。

【図101】



【図102】



フロントページの続き

(71)出願人 596016535

2011 North Shoreline
Boulevard Mountain
View, California 94039
U. S. A.

(72)発明者 ティモシー・ジェイ・バン フック
アメリカ合衆国 カリフォルニア州94027,
アサートン, オーク・グローブ・アベニ
ュー, 224番

(72)発明者 ハワード・ハオ・チェン
アメリカ合衆国 ヴァージニア州98052,
レドモンド, 150ス・アベニュー, エヌ
イー, 4820番

(72)発明者 アンソニー・ビー・ディロウリア
アメリカ合衆国 カリフォルニア州94086,
サニーベイル, エスカロン・アベニ
ュー, 1000番, M-1100号室

(72)発明者 カロル・フィリップ・ゴセット
アメリカ合衆国 カリフォルニア州94043,
マウンテン・ビュー, バーゴイン, 1169番

(72)発明者 ロバート・ジョーダン・ムーア
アメリカ合衆国 フロリダ州32779, ロン
グウッド, ヴィレッジ・ビュー・レイ
ン, 436番

(72)発明者 ステファン・ジェイ・シェバード
アメリカ合衆国 カリフォルニア州95014,
クベルティノ, ヴァレー・グリーン・ドラ
イブ, 20990番, 671号室

(72)発明者 ハロルド・ステファン・アンダーソン
アメリカ合衆国 カリフォルニア州95037,
モーガン・ヒル, ヴィスタ・デル・ヴァ
ル・コート, 16750番

(72)発明者 ジョン・ブリンセン
アメリカ合衆国 カリフォルニア州95124,
サン・ホセ, ロレンツェン・ドライブ,
1729番

(72)発明者 ジェフリー・チャンドラー・ドウティー
アメリカ合衆国 カリフォルニア州94301,
バロ・アルト, ウィルソン・ストリート,
1295番

(72)発明者 ネイサン・エフ・ブーリー
アメリカ合衆国 カリフォルニア州94041,
マウンテイン・ビュー、シルヴァン・アベ
ニュー、680番、12号室

(72)発明者 バイロン・シェバード
アメリカ合衆国 カリフォルニア州95060,
サンタ・クルーズ、ウッドロー・アベニュー、504番

(72)発明者 竹田 玄洋
京都府京都市東山区福稲上高松町60番地
任天堂株式会社内

(72)発明者 加藤 周平
京都府京都市東山区福稲上高松町60番地
任天堂株式会社内

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-325759

(43)Date of publication of application : 16.12.1997

(51)Int.Cl. G09G 5/36

A63F 9/22

G06T 11/00

G10K 15/04

(21)Application number : 08-352115 (71)Applicant : NINTENDO CO LTD
SILICON GRAPHICS INC

(22)Date of filing : 22.11.1996 (72)Inventor : VAN HOOK TIMOTHY J
HOWARD HAO CHENG
ANTHONY P DELAURIER
GOSSETT CARROLL PHILIP
MOORE ROBERT JORDAN
STEPHEN J SHEPARD
HAROLD STEPHEN
ANDERSON
PRINCEN JOHN
DOUGHTY JEFFREY
CHANDLER
POOLEY NATHAN F
BYRON SHEPPARD
TAKEDA GENYO
KATO SHUHEI

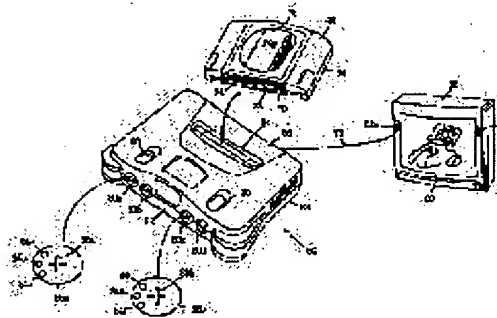
(30)Priority

Priority number : 95 561718 Priority date : 22.11.1995 Priority country : US

(54) HIGH PERFORMANCE LOW COST VIDEO GAME SYSTEM PROVIDED WITH COPROCESSOR PROVIDING HIGH SPEED HIGH EFFICIENCY 3D GRAPHICS AND DIGITAL SOUND SIGNALS PROCESSING

(57)Abstract:

PROBLEM TO BE SOLVED: To model the world with 3D and to make possible projecting that model on a two-dimensional view plane selected based on a revisable viewpoint by making a video game storage a shape of a detachable memory cartridge capable of inserting into a slot.



SOLUTION: The video game storage 54 is inserted into the slot 64 of a main unit. In such a manner, a read only memory 76 is connected to the electronic circuit of the main unit through a printed circuit board 70 and related edge points of contact 74. Then, a user operates a source switch 88 to turn a video

game system 50 on. Thus, the main unit 52 starts a play of a video game based on software stored in the read only memory 76. Then, the user operates a controller 86, and imparts an input to the main unit 52 to execute the play of the video game.

LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's

decision of rejection]

[Date of requesting appeal against
examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

*** NOTICES ***

**Japan Patent Office is not responsible for any
damages caused by the use of this translation.**

1.This document has been translated by computer. So the translation may not reflect the original precisely.

2.**** shows the word which can not be translated.

3.In the drawings, any words are not translated.

CLAIMS

[Claim(s)]

[Claim 1] The interactive video game system characterized by providing the following Interactive user input equipment The main processor which is connected to the aforementioned input unit, has an address space, and chooses a view as a dialogue according to the input from the aforementioned user input equipment By connecting with the aforementioned main processor and projecting the polygon showing a 3-dimensional world on a two-dimensional viewing flat surface In order to generate image data to a dialogue according to the selected view, the predetermined graphics feature set is offered. It is used by a graphics function and the audio processing facility in common at least. In order to have the vector unit which can perform the Scala unit and two or more calculation in parallel, to have the microcode storage which memorizes a microcode and to perform a graphics function and the audio processing facility concerned concerned The signal processor which performs the microcode concerned in the microcode storage concerned The 1st portion can memorize the texture map by which a Color Index is not carried out to the texture map by which the Color Index was carried out. the 2nd portion It has the texture memory which can memorize the color look-up table for a texture map and/or the texture maps by which the Color Index was carried out. While giving every cycle

1-pixel mode or every cycle 2-pixel mode and making hardware into the minimum The display processor containing the display pipeline hardware which gives the abundant feature sets containing the level of detailed processing, A video interface, a voice interface, and a serial interface, The co-processor containing a parallel peripheral interface adapter, and the aforementioned signal processor, Each of the aforementioned display processor, the aforementioned video interface, the aforementioned voice interface, the aforementioned serial interface, and the aforementioned parallel peripheral interface adapter Have the circuit which accesses main memory and it connects with the aforementioned co-processor through a 9-bit wide bus. The instruction which gives an address space common to the aforementioned co-processor and the aforementioned main processor, and is executed by the aforementioned main memory further at least, A color frame buffer, a depth buffer, and a graphic microcode, A speech processing microcode, at least one display list, and at least one texture map, The main memory which memorizes the data structure of at least one audio output buffer, The video signal generation circuit which generates the video signal which is connected to the aforementioned video interface of the aforementioned co-processor, and is displayed on a color television receiver, Housing, a security chip, read-only memory, and the storage that has at least one another memory apparatus and that can be removed, The aforementioned co-processor equipped with the structure which maps read-only memory and the another memory apparatus concerned concerned in the address space of the aforementioned main processor, The aforementioned read only memory which memorizes the aforementioned graphics and an audio processing microcode in early stages, The connector which connects the aforementioned co-processor to the aforementioned storage which can be removed, The processor which is connected to the serial interface of the aforementioned co-processor, and performs a serial interface function and a security function, And the serial peripheral-interface-adapter circuit which is equipped with the boot ROM which gives a main processor initial-program-load instruction, and is connected to the security chip of the aforementioned storage which can be removed through the aforementioned connector [Claim 2] The graphics display system real-time [interactive] characterized by providing the following At least one user input equipment The main RAM which gives a common address space The main processor which is connected in order to carry out addressing of the aforementioned main memory, and is connected also to the aforementioned user input equipment, answers the input inputted from the aforementioned user input equipment on real time, and memorizes an instruction to

the aforementioned main memory, executes the instruction from the aforementioned main memory, and memorizes the display list of a graphics command, and the reproduction list of audio commands to the aforementioned main memory at least Connect in order to carry out addressing of the aforementioned main memory, and the microcode memorized by the aforementioned main memory is taken out and performed. The graphics command of the aforementioned display list and the audio command of the aforementioned reproduction list are read from the aforementioned main memory. Answer the audio command of the aforementioned reproduction list and audio sample data is generated. And the signal processor which memorizes the aforementioned sample data to the audio output buffer which answered the aforementioned display list, generated the graphics viewing command, and was assigned in the aforementioned main memory, Connect in order to carry out addressing of the aforementioned main memory, and it is partially based on at least one texture map and other graphics data which were memorized by the aforementioned main memory at least, and image data is generated. The display processor which answers a graphics viewing command, creates the image data concerned, and memorizes the image data concerned in the color picture frame buffer in the aforementioned main memory, Connect in order to carry out addressing of the aforementioned main memory, and it synchronizes with a display raster scan. The video interface which reads the aforementioned color picture frame buffer, and the audio interface which is connected in order to carry out addressing of the aforementioned main memory, and reads the aforementioned audio output buffer synchronizing with real-time sound generation

[Claim 3] How to operate the graphics display system which has the video signal generation structure which generates the main processor characterized by providing the following, the co-processor connected to the main processor concerned, the main RAM which is connected to the co-processor concerned and can carry out addressing by a main processor and the co-processor concerned concerned, and the video signal for a display (a) The step which memorizes a main processor code to the aforementioned main memory (b) The step which performs the aforementioned main processor code memorized by the aforementioned step which carries out storage including a co-processor code, a task list, at least one texture map, and the step that memorizes a color look-up table to the aforementioned main memory by the aforementioned main processor (c) The step which takes out the aforementioned task list from the aforementioned main memory (d) In the aforementioned co-processor code memorized by the aforementioned step (b) Follow partially at least and the

aforementioned task list is processed by the aforementioned co-processor. The following (1) aforementioned texture maps and aforementioned color look-up tables are loaded to on-chip texture memory from the aforementioned main memory, (2) At least one 3-dimensional geometry conversion is performed about 1 set of peaks using a vector unit including performing two or more calculation in parallel with the Scala unit and a vector unit, (3) A triangular command is generated based on the aforementioned 3-dimensional geometry conversion, (4) Answering the aforementioned triangular command and generating a pixel value and the (5) aforementioned texture memory are accessed twice. A Color Index theque cell is outputted based on the aforementioned triangular command, (6) A compound pixel value is generated combining the aforementioned theque cell and the generated pixel value, (7) with at least one pixel value memorized in accessing the pixel value in the frame buffer memorized by the aforementioned main memory, and the (8) aforementioned frame buffer Based on the comparison using the depth buffer memorized by the (9) aforementioned main memory, the aforementioned compound pixel value is conditionally written [blending the aforementioned compound pixel value,] in the aforementioned frame buffer, (10) An output audio sample is generated using the aforementioned Scala unit and the aforementioned vector unit including performing two or more calculation in parallel with the aforementioned vector unit, (11) The aforementioned output audio sample is memorized to the aforementioned main memory, The step which reads the aforementioned frame buffer on real time synchronizing with a processing step also including execution of a **** step, and the scan of the (e) color television receiver, and changes the content of the aforementioned frame buffer into a composite video signal, (f) Step which reads the output audio sample by which storage was carried out [aforementioned] on real time, and changes the aforementioned audio sample into stereo sound

[Claim 4] The process which generates at least one display-mode control command for processing according to a 3-dimensional graphics system characterized by providing the following The set mode command concerned is the command identifier field including the 6-bit binary value of 101111 including the step which generates at least one set mode command. the following modal-control field, i.e., (k), -- before reading the following primitive, it specifies whether a primitive is made to write in a frame buffer -- Choose the atomic primitive mode field and (i) display pipeline cycle control mode. Enable alternatively the cycle type mode field and (h) perspective texture correction. Enable alternatively detailed processing of the perspective texture ***** mode field and the (g) texture. Make it possible alternatively to make

clear the texture detailed mode field and the (f) texture. Enable alternatively processing of the detailed level value of texture clear ***** mode and the (e) texture. Make it possible to search a texture value alternatively from texture detailed ***** mode and the (d) color look-up table. Specify the type of the theque cell in the ***** look-up table mode field and the (c) color look-up table. Specify how the texture look-up table TAIPUMODO field and (b) theque cell should be sampled. Specify whether the sample type mode field and (a) theque cell should be filtered using 2x2 half theque cell interpolation. Specify whether the MIDDOTEKU cell mode field and (Z) texture filter should interpolate the theque cell in the pipeline cycle 0 in bilinear. Specify whether the 1st BAIRAPU mode field and (Y) texture filter should interpolate the theque cell in the pipeline cycle 1 in bilinear. Specify whether color conversion of the theque cell outputted from a texture filter between the 2nd BAIRAPU mode field and (X) pipeline cycle 0 should be carried out. Enable alternatively the theque cell translation-mode field and (W) chroma-key board input. Choose the type of the chroma-key ***** mode field and RGB (V2) dithering. Choose the type of the RGB dither selection mode field and alpha (V1) dithering. Specify the alpha dither selection mode field and (V) blender parameter. Specify whether two or more blend mode WORD and the (M) aforementioned blender should be made into a compulsive enable. Specify whether it should be used in order that the compulsive blend ***** mode field and the (L) scope may determine pixel alpha. Specify whether it should be used in order that the scope multiplication was carried out [the scope] by the alpha scope selection mode field and (K) alpha may determine pixel alpha and a scope. Specify the scope Times alpha selection mode field and (J) Z buffering. The scope destination mode field which specifies Z mode selection mode field and (I) scope destination, (H) Specify whether it should be updated only in the place which the color protruded from the scope. Make possible alternatively the color mode field of a scope, the (G) color, and/or frame buffer memory access scope read-out / correction / write-in. Enable Z-uffer writing alternatively by whether the picture read-out ***** mode field and (F) color writing are enabled. In Z update ***** mode field and (E) depth comparison, specify possible conditional color writing. Permit blend ***** using Z comparison ***** mode field and the (D) scope. Choose between the ANCHIERIASU ***** mode field, (C) primitive depth, and pixel depth. Specify whether Z source selection mode field and (B) random noise should be used for alpha comparison. the alpha comparison ***** mode field which is on the DIZAARUFA ***** mode field and (A) alpha comparison, and enables conditional color writing, and ** -- at least one

[Claim 5] The process containing the step which generates the cycle type mode which chooses 1 cycle mode for every pixel, the two-cycle mode for every pixel, copy mode, and the display pipeline cycle control mode in fill mode according to claim 4.

[Claim 6] (1) storage of TEKUSERU in the color look-up table in the brightness alpha format which supplies storage, (2) 8-bit brightness value, and the 8-bit alpha value of TEKUSERU in the color look-up table in 5-bit red, 5 bits green, 5 bits blue, and the 1-bit RGBA format of alpha, and ** -- the process containing the step which generates the texture lookup TAIPUMODO field which chooses either inside according to claim 4

[Claim 7] (1) a point sampling, a (2)2x2 array sampling, and ** -- the process containing the step which generates the sample type mode field which chooses inner either according to claim 4

[Claim 8] (1) the process containing the step which generates a magic square matrix, (2) BEIYA matrix, (3) noises, (4) dithering nothing, and the RGB dither selection mode field that is alike and chooses either of the based dithering according to claim 4

[Claim 9] (1) The process containing the step which generates the alpha dithering selection mode field which specifies a predetermined pattern, the negative of a predetermined pattern and (2) (3) noise, and dithering based on those without (4) dithering according to claim 4.

[Claim 10] Alternative **** of the 1st blender input between the pipeline cycles 0, Alternative increase of the 1st blender input between the pipeline cycles 1, Alternative increase of the 2nd blender input between the pipeline cycles 0, Alternative increase of the 2nd blender input between the pipeline cycles 1, Alternative increase of the 3rd blender input between the pipeline cycles 0, Alternative increase of the 3rd blender input between the pipeline cycles 1, The process according to claim 4 which contains the step which generates two or more blend mode WORD which specifies the blender parameter which specifies alternative increase of the 4th blender input between alternative increase of the 4th blender input, and the pipeline cycle 1 between the pipeline cycles 0.

[Claim 11] (1) the operation to a clamp, (2) laps, and all (3) scopes, (4) preservation, and ** -- the process containing the step which generates the scope destination mode field which chooses inner one of scope destination modes according to claim 4

[Claim 12] (1) opacity, (2) osmosis, (3) transparence, (4) DEKARU, and ** -- the process containing the step which generates Z mode selection mode field which chooses inner one of Z-uffer ring modes according to claim 4

[Claim 13] The system which generates at least one three-dimensional display

modal-control command for processing according to a 3-dimensional graphics system characterized by providing the following At least one processor At least one memory The circuit combined with the aforementioned processor and the aforementioned memory in order to supply at least one set mode command ***** and the aforementioned set mode command are the 6-bit binary value of 101111.

[Claim 14] A system including a means to generate the cycle type mode which chooses 1 cycle mode for every pixel, the two-cycle mode for every pixel, copy mode, and the display pipeline cycle control mode in fill mode according to claim 13.

[Claim 15] (1) storage of the theque cell in the color look-up table in the brightness alpha format which supplies storage, (2) 8-bit brightness value, and the 8-bit alpha value of the theque cell in the color look-up table in 5-bit red, 5 bits green, 5 bits blue, and the 1-bit RGBA format of alpha, and ** -- a system including a means to supply the texture lookup TAIPUMODO field which chooses either inside according to claim 13

[Claim 16] (1) a point sampling, a (2)2x2 array sampling, and ** -- a system including the circuit which supplies the sample type mode field which chooses inner either according to claim 13

[Claim 17] (1) a system including the circuit which supplies a magic square matrix, (2) BEIYA matrix, (3) noises, (4) dithering nothing, and the RGB dither selection mode field that is alike and chooses either of the based dithering according to claim 13

[Claim 18] (1) A system including a means to generate the alpha dithering selection mode field which specifies a predetermined pattern, the negative of a predetermined pattern and (2) (3) noise, and dithering based on those without (4) dithering according to claim 13.

[Claim 19] Alternative increase of the 1st blender input between the pipeline cycles 0, Alternative increase of the 1st blender input between the pipeline cycles 1, Alternative increase of the 2nd blender input between the pipeline cycles 0, Alternative increase of the 2nd blender input between the pipeline cycles 1, Alternative increase of the 3rd blender input between the pipeline cycles 0, Alternative increase of the 3rd blender input between the pipeline cycles 1, The system according to claim 13 which includes a means to generate two or more blend mode WORD which specifies the blender parameter which specifies alternative increase of the 4th blender input, between alternative increase of the 4th blender input, and the pipeline cycle 1 between the pipeline cycles 0.

[Claim 20] (1) the operation to a clamp, (2) laps, and all (3) scopes, (4) preservation, and ** -- a system including the circuit which generates the scope destination mode

field which chooses inner one of scope destination modes according to claim 13

[Claim 21] (1) opacity, (2) osmosis, (3) transparence, (4) DEKARU, and ** -- a system including the circuit which supplies Z mode selection mode field which chooses inner one of Z-uffer ring modes according to claim 13

[Claim 22] The 3-dimensional GURAFIKU system characterized by providing the following The step which interprets the command identifier field which is the process which interprets at least one set mode command, and includes the 6-bit binary value of (1) 101111 (2) the following modal-control field -- getting it blocked -- (k) -- before reading the following primitive, it specifies whether a primitive is made to write in a frame buffer -- Choose the atomic primitive mode field and (i) display pipeline cycle control mode. Enable alternatively the cycle type mode field and (h) perspective texture correction. Enable alternatively detailed processing of the perspective texture ***** mode field and the (g) texture. Make it possible alternatively to make clear the texture detailed mode field and the (f) texture. Enable alternatively processing of the detailed level value of texture clear ***** mode and the (e) texture. Make it possible to search a texture value alternatively from texture detailed ***** mode and the (d) color look-up table. Specify the type of TEKUSERU in the ***** look-up table mode field and the (c) color look-up table. Specify how the texture look-up table TAIPUMODO field and (b) TEKUSERU should be sampled. Specify whether the sample type mode field and (a) TEKUSERU should be filtered using 2x2 half TEKUSERU interpolation. Specify whether the MIDDOTEKU cell mode field and (Z) texture filter should interpolate TEKUSERU in the pipeline cycle 0 in . bilinear. Specify whether the 1st BAIRAPU mode field and (Y) texture filter should interpolate TEKUSERU in the pipeline cycle 1 in bilinear. Specify whether color conversion of TEKUSERU outputted from a texture filter between the 2nd BAIRAPU mode field and (X) pipeline cycle 0 should be carried out. Enable alternatively the TEKUSERU translation-mode field and (W) chroma-key board input. Choose the type of the chroma-key ***** mode field and RGB (V2) dithering. Choose the type of the RGB dither selection mode field and alpha (V1) dithering. Specify the alpha dither selection mode field and (V) blender parameter. Specify whether two or more blend mode WORD and the (M) aforementioned blender should be made into a compulsive enable. Specify whether it should be used in order that the compulsive blend ***** mode field and the (L) scope may determine pixel alpha. Specify whether it should be used in order that the scope which increased by the alpha scope selection mode field and (K) alpha may determine pixel alpha and a scope. Specify the scope Times alpha selection mode field and (J) Z buffering. The scope destination

mode field which specifies Z mode selection mode field and (I) scope destination, (H) Specify whether it should be updated only in the place which the color protruded from the scope. Make possible alternatively the color mode field of a scope, the (G) color, and/or frame buffer memory access scope read-out / correction / write-in. Enable Z-uuffer writing alternatively by whether the picture read-out ***** mode field and (F) color writing are enabled. In Z update ***** mode field and (E) depth comparison, specify possible conditional color writing. Permit blend ***** using Z comparison ***** mode field and the (D) scope. Choose between the ANCHIERIASU ***** mode field, (C) primitive depth, and pixel depth. Specify whether Z source selection mode field and (B) random noise should be used for alpha comparison. the alpha comparison ***** mode field which is on the DIZAARUFA ***** mode field and (A) alpha comparison, and enables conditional color writing, and ** -- with the step which interprets at least one (3) Step which generates a picture based on the aforementioned step (2) partially at least

[Claim 23] The process containing the step which interprets the cycle type mode which chooses 1 cycle mode for every pixel, the two-cycle mode for every pixel, copy mode, and the display pipeline cycle control mode in fill mode according to claim 22.

[Claim 24] (1) storage of the theque cell in the color look-up table in the brightness alpha format which supplies storage, (2) 8-bit brightness value, and the 8-bit alpha value of the theque cell in the color look-up table in 5-bit red, 5 bits green, 5 bits blue, and the 1-bit RGBA format of alpha, and ** -- the process containing the step which interprets the texture lookup TAIPUMODO field which chooses either inside according to claim 22

[Claim 25] (1) a point sampling, a (2)2x2 array sampling, and ** -- the process containing the step which interprets the sample type mode field which chooses inner either according to claim 22

[Claim 26] (1) the process containing the step which interprets a magic square matrix, (2) BEIYA matrix, (3) noises, (4) dithering nothing, and the RGB dither selection mode field that is alike and chooses either of the based dithering according to claim 22

[Claim 27] (1) The process containing the step which interprets the alpha dithering selection mode field which specifies a predetermined pattern, the negative of a predetermined pattern and (2) (3) noise, and dithering based on those without (4) dithering according to claim 22.

[Claim 28] The 1st blender input is alternatively increased between the pipeline cycles 0. Alternative increase of the 1st blender input between the pipeline cycles 1, Alternative increase of the 2nd blender input between the pipeline cycles 0,

Alternative increase of the 2nd blender input between the pipeline cycles 1,
Alternative increase of the 3rd blender input between the pipeline cycles 0,
Alternative increase of the 3rd blender input between the pipeline cycles 1,
Alternative increase of the 4th blender input between the pipeline cycles 0, The
process according to claim 22 which contains the step which interprets two or more
blend mode WORD which specifies the blender parameter which specifies alternative
increase of the 4th blender input between the pipeline cycles 1.

[Claim 29] (1) the operation to a clamp, (2) laps, and all (3) scopes, (4) preservation,
and ** -- the process containing the step which interprets the scope destination
mode field which chooses inner one of scope destination modes according to claim 22

[Claim 30] (1) opacity, (2) osmosis, (3) transparence, (4) DEKARU, and ** -- the
process containing the step which interprets Z mode selection mode field which
chooses inner one of Z-uffer ring modes according to claim 22

Since it became timeout time, translation result display processing is stopped.

*** NOTICES ***

Japan Patent Office is not responsible for any
damages caused by the use of this translation.

1.This document has been translated by computer. So the translation may not reflect
the original precisely.

2.**** shows the word which can not be translated.

3.In the drawings, any words are not translated.

DETAILED DESCRIPTION

[Detailed Description of the Invention]

[0001]

[The technical field to which invention belongs] this invention relates to the video
game system of a low cost. this invention relates to the video game system which can
project the model concerned on the two-dimensional view flat surface chosen based
on the view which can model a world by three dimensions and can be changed more at
a detail.

[0002] The imaginative power of the background of this invention and [summary] human being is stimulated by the visual-sense picture. All these scenes that are not forgotten, such as **** which he pictures to himself when a ***** thing and a novel are actually read at the thing and night which are seen at the time of sunset, consist of visual-sense pictures. Everywhere in historical, people are a pencil or paints or tried to record these pictures on videotape. However, people can begin generate those pictures now with the advent of a computer with the same clearness as that as which the picture was expressed the actual world or in imaginative power, details, and an actual feeling.

[0003] A home video game machine of the computer base like the Nintendo entertainment system and the Super Nintendo entertainment system has stored a success very much from exciting video graphics being interactively generable. However, without needing additional extension hardware, generally these conventional video graphic systems operate by two-dimensional, are the methods which were like stopping a clipping of flat paper to a bulletin board somewhat, and generate graphical display by flat (flat surface) picture expression. Although a very exciting game play can generate using the two-dimensional graphic method, 2D system cannot offer the realism realized by the 3-dimensional graphic system.

[0004] 3D graphic system differs from 2D graphics fundamentally. A "world" is expressed with the 3D graphic method in 3-dimensional space. This system enables a user to choose a view in a world. What this system "projects for" a world based on the selected view generates a picture. This result is the true 3-dimensional picture equipped with depth and realism.

[0005] In order [of such a real 3D picture, for example, a vehicle, that an expert is not trusted over many years, an airplane, and a molecule] overly to generate the dinosaur of a minute model (virtual reality which was seen from the cockpit of a jet fighter, or the front sheet of the bobsled of the Olympic Games), and a "Jura chic park", the supercomputer and the high-end workstation were used. However, the computer system needed costs (this exceeds at least the range which an average customer's hand reaches) of tens of thousands of dollars until now, in order to generate such a picture interactively.

[0006] The low-cost highly efficient 3D graphic system currently indicated here has the intention of giving an opportunity having a dialog inside wonderful 3D world to the game player of not only an expert but a large number for the first time by the fully characteristic low-cost highly efficient system. It is really several times as wonderful more realistic 3-dimensional animation as the power of what home computer system,

and the great graphics which a player obtains. These all are supplied at cost low enough, as it is within limits which an average customer's hand reaches.

[0007] Some examples of many advantageous features realized by the system by this invention below are shown.

- In order to offer the video game play and other graphics applications of realistic dialogue 3D graphics and a low-cost system of a low-price system and/or to generate the specific screen effect, In the co-processor and low-cost color-television standard system which offers the optimal feature set / architecture, the highly efficient 3D graphics, and the digital acoustical treatment for low-price systems which are used for a color television receiver Realize quality stereo sound and 3D graphics. The technique / composition, and storage with which the signal processor and the integrated RAM method shared by graphics digital processing and audio signal processing increasing flexibility and all the main system element communicating through common use RAM and narrow main memory bus width of face are compensated (for example) The code which can be performed from a pocket memory cartridge can load to common RAM. The graphic co-processor which can be loaded to that it can access by the main processor through co-processor memory access / coupled circuit and microcode storage A graphics function and an audio function are called in that - microcode which receives a microcode from a pocket storage is loaded by execution of a "boot ROM" instruction in order to give the further flexibility and to simplify the trouble of compatibility, and a co-processor. And in order to give an interface between a graphics coprocessor and other portions of a system That the optimal command and a related format are used, the definition of a specific hardware register, - vector unit which offers highly efficient operation with effective graphics of the set microcode of a co-processor register, and audio composition / processing in which it has a format and the function of relation Giving optimum performance to the graphics in a low-cost package and audio digital processing and a pipeline rasterizing engine A pin output with few offering every cycle 1-pixel mode and every cycle 2-pixel mode, and making hardware cost into the minimum, and offering the abundant feature sets and co-processors [0008] [Explanation with suitable detailed example] drawing 1 shows the example of the video game system 50 concerning this invention. The video game system 50 of this example consists of a main unit 52, video game storage 54, and a hand hold controller 56 (or other user input equipments). The main unit 52 is connected to the usual home color television receiver 58 in this example. A color television receiver 58 displays 3D video game picture on the television screen, and reproduces stereo sound by the loudspeaker 62.

[0009] In this example, the video game storage 54 has taken the form of the memory cartridge which can be inserted in the slot 64 prepared in the upper surface 66 of main YUNITTO 52 and which can be removed. The video game storage 54 consists of housing 68 of a plastic which contained the read-only memory (ROM) chip 76 inside. Read-only memory 76 stores video game software in this example. If the video game storage 54 is inserted in the main unit slot 64, electric contact 74 of a cartridge will gear with electric contact of the "edge connector" in a main unit. Thereby, the read-only memory 76 in the video game storage 54 is electrically connected to the electronic circuitry in a main unit 52.

[0010] "Read-only memory" chip 76 has memorized the software instruction and other information about specific video game. The RO memory chip 76 in the video game storage 54 stores the instruction for adventure games, and other information. The RO memory chip 76 in another video game storage 54 stores the instruction for carrying out a driving game, i.e., a car race game, and information. Furthermore, the RO memory chip 76 in another video game storage 54 stores the instruction for carrying out an educational game, and information. In order to completely carry out another game, thereby, the RO memory chip 76 (and the storage of other arbitrary circuits is also contained) in the video game storage 54 is connected to a main unit 52 that the user of the video game system 50 should just insert the only suitable video game storage 54 in the main unit slot 64. It enables a main unit 52 to access by this the information stored in the RO memory chip 76. This information controls a main unit 52 to perform suitable video game by displaying the specified picture on a color television receiver 58, and reproducing the sound under control of the video game software in read-only memory.

[0011] In order to carry out video game by the video game system 50, a user connects a main unit 52 to a color television receiver 58 by connecting a cable 78 first. A main unit 52 outputs "video" signal and the "audio" signal which control a color television receiver 58. "Video" signal controls the picture displayed on the television screen 60, and, on the other hand, an "audio" signal is reproduced by the television loudspeaker 62 as a sound. It is required to use further the unit "RF modulator" between the lines of a main unit 52 and a color television receiver 58 depending on the form of a color television receiver 58. "RF modulator" (not shown) changes the video and the audio output of a main unit 52 into the television signal of a broadcast formula. It is received by the "tuner" inside a television television machine, and this television signal is processed.

[0012] It is also required that a user should connect a main unit 52 to a power supply.

This power supply is inserted in the electric socket of standard home use, and builds in the usual AC adapter (not shown) which transforms a home power supply into the low DC voltage suitable for operating a main unit 52.

[0013] Next, a user connects the hand controllers 56a and 56b to the connector 80 to which it corresponds on the front panel 82 of a main unit. The hand controller 56 can take various kinds of form. The controller 56 shown in this example is equipped with a push button 84 and a direction switch, or other control 86. Since the direction (a top, the bottom, left, or right) to which the character displayed for example, on the television screen 60 should move is pinpointed, and/or since the view in 3D world is specified, the direction switch 86 is used. A joy stick, mouse pointer control, and other conventional user input equipments are contained in other possible things. In this example, a controller 56 is connectable with a main unit 52 to four, in order to make the game of a four-person play possible.

[0014] Next, a user chooses the video game storage 54 which stores the video game which he wants to enjoy, and inserts the video game storage 54 in the slot 64 of a main unit (this connects read-only memory 76 to the electronic circuitry of a main unit through a printed circuit board 70 and the related edge contact 74). Next, a user operates an electric power switch 88 and turns ON the video game system 50. Thereby, a main unit 52 starts the play of video game based on the software stored in read-only memory 76. A user operates a controller 86, gives an input to a main unit 52, and performs the play of video game. For example, a game is started when a user pushes one of the push buttons 84. As mentioned above, by operating the direction switch 86, the character of animation moves in the different direction on the television screen 60, or the view of the user in 3D world changes. Depending on the specific video game memorized in the video game storage 54, the various control 84 and 86 on a controller 56 can perform the function which is different at time to be different. When a user wants to carry out the restart of the play of a game, a user can push a reset button 90.

*** NOTICES ***

Japan Patent Office is not responsible for any damages caused by the use of this translation.

1.This document has been translated by computer. So the translation may not reflect the original precisely.

2.**** shows the word which can not be translated.

3.In the drawings, any words are not translated.

EXAMPLE

[0015] The [example of 3D screen effect] video game system 50 can process the model of digital display or a 3-dimensional world on real time to a dialogue so that a world (or the part) may be displayed from the arbitrary views in this world. For example, the video game system 50 answers real time input from the game controller 86, and can change a view into a dialogue. For this reason, all the places ordered that the inside of a world is moved and a game player goes for example, are seen, it can let the view of "the person of imagination" of going there pass, and a game player can see a world. This capacity that displays the property of 3D picture on a dialogue on real time is ***** about a very real and exciting game play.

[0016] drawing 2 (a) – drawing 3 (f) -- the video game system 50 -- the screen top of a color television receiver 58 -- ***** -- one mere example of the 3-dimensional screen effect which can do things is shown Since patent documents cannot print in a color, although drawing 2 (a) – drawing 3 (f) are shown by black and white, the video game system 50 can display these different screens in a bright color on a color television receiver. furthermore, the video game system 50 -- operation of the game controller 86 -- answering -- these pictures -- real time -- very -- quick in for example, several seconds or dozens of seconds -- ***** -- things are made

[0017] Each of drawing 2 (a) – drawing 3 (f) is generated using 3 dimensional models of the "world" expressing the castle which is on the top of a hill. This model consists of "textures" (picture memorized by digital one) "is mapped" on the shaped surface decided by a geometry (getting it blocked polygon) and this geometry. the video game system 50 -- suitable -- the size of these geometries -- deciding -- rotating -- moving -- them -- " -- the realistic picture of the 3-dimensional world which projected, unified " and them and was seen from arbitrary views -- ***** The video game system 50 answers operation of the user by the game controller 86, and can perform this on real time to a dialogue.

[0018] Drawing 2 (a) – drawing 2 (c), and drawing 3 (f) show drawing from the empty of the castle seen from four different views. I want to note that each drawing is a perspective diagram. The video game system 50 has a little delay that there is no delay which can be recognized, drawing (and drawing between them) from these empty

can be interactively generated in several seconds, and for the reason, drawing from these empty appears as if the video game player was actually flying over the castle. [0019] Drawing 2 (d) and drawing 2 (e) show drawing which is the main gate of a castle or was looked up at from the ground near it. The video game system 50 answers the input of the game controller which "orders to land in front of a castle" a view, and orders "the observer of imagination" (person of imagination who moves in 3D world that get it blocked and a screen is displayed through the person's eyes), suitable in another direction, and generates these drawings on real time in a dialogue. Drawing 2 (d) shows the example of "texture mapping" that the texture (picture) of a brick wall is mapped very much by the wall of a castle in a realistic image at a ***** sake.

[0020] It is shown that the electronic circuitries in a main unit 52 with main [video game system whole electronics] drawing 4 have a main processor 100, a co-processor 200, and main memory 300. A main processor 100 is a computer which makes the video game program supplied to the video game storage 54 run based on the input given by the controller 56. A co-processor 200 generates a picture and sound based on the instruction and command which were obtained from the main processor 100. Main memory 300 is high speed storage which memorizes information required for a main processor 100 and a co-processor 200 to operate, and is shared by the main processor 100 and the co-processor 200. In this example, all accesses to main memory 300 are performed through a co-processor 200.

[0021] In this example, a main processor 100 accesses a video game program through a co-processor 200 with the communication path 102 between a main processor 100 and a co-processor 200. A main processor 100 can be read from the video game storage 54 through another communication path 104 between a co-processor 200 and the video game storage 54. A main processor 100 can copy a video game program to main memory 300 from the video game storage 54 with a path 106, and, next, can access the video game program in main memory 300 through a co-processor 200 and paths 102 and 106.

[0022] A main processor 100 sometimes generates the list of commands which direct what should be accomplished to a co-processor 200. In this example, the co-processor 200 is equipped with the specific integrated circuit (ASTC) which is the exclusive highly efficient application which has the optimized internal design, in order to process 3D graphics and a digital audio quickly. A co-processor 200 answers the command given by the main processor 100 through the path 102, and generates the video and the audio signal which are supplied to a color television receiver 58. A co-processor 200 uses the graphics data memorized by main memory 300 and/or the

video game storage 54, audio data, and other data, and generates a picture and sound. [0023] In this example, as for drawing 4, the co-processor 200 shows that it has the signal processor 400 and the display processor 500. A signal processor 400 performs and embeds geometric processing of graphics, and processing of a digital audio signal under control of the "microcode" computer program supplied from the video game storage 54, and is a programmable microcontroller. The display processor 500 gives a graphics primitive and, thereby, is a ***** high-speed state machine about the picture displayed on a color television receiver 58. Although a signal processor 400 and the display processor 500 operate separately, a signal processor 400 can supervise the display processor 500 by sending a graphics command. Both the signal processor 400 and the display processor 500 can be controlled directly by the main processor 100. The following is the example of the function and operation which can perform a signal processor 400 and the display processor 500.

[0024] The combination and the blend fogging anti-aliasing frame buffer of application of the coordinate generating and the texture of the command generating display processor raster rye ZESHON texture of signal processor matrix control, 3D conversion and lighting . clipping, perspective, and application / display processor of a view port, and a filtering and a color, and frame buffer control [0025] Drawing 5 shows the main processings performed by the main processor 100 in the video game system 50 of this example, a co-processor 200, and main memory 300. A main processor 100 receives an input from the game controller 56, performs the video game program given by the video game storage 54, and processes a game (block 120). This gives animation and assembles the graphics and audio command which are used by the co-processor 200. The graphics and sound command which were generated by the main processor 100 are processed by blocks 122, 124, and 126 (these each is performed by the co-processor 200). In this example, a signal processor 400 performs 3D geometrical conversion and lighting processing (block 122), and generates the graphics viewing command to the display processor 500. the display processor 500 -- the primitive (for example, a line, a triangle, and a rectangle) of graphics -- " -- the picture which is drawn and is displayed on " and a color television receiver 58 -- ***** If the display processor 500 is still more nearly required by "rasterizing" each primitive, a this "drawing" function or a rendering function will be performed by sticking a texture on this (block 126). The display processor 500 performs this processing with the speed of millions "a pixel (element of the picture of color television)" very quickly, for example, per second. The display processor 500 writes the picture output in the frame buffer in main memory 300 (block 128). This frame buffer memorizes the digital display

of the picture which should be displayed on the television screen 60. Another circuit in a co-processor 200 reads information in this frame buffer, and in order to display this on a color television receiver 58, it carries out attitude (block 130).

[0026] A signal processor 400 also processes the sound command inputted from the main processor 100 again using a digital audio signal approach (block 124). A signal processor 400 writes a digital audio output in the sound buffer in main memory 300. Main memory 300 carries out "the buffer (that is, storage)" of this sound output temporarily (block 132). Other circuits in a co-processor 200 read in main memory 300 this sound data by which the buffer was carried out, change this into an electric audio signal (the stereo left and right channel), and it is sent to the television loudspeakers 62a and 62b, and they reproduce it (block 134).

[0027] A color television receiver 58 displays per second 30 or the new picture of 60 sheets. This "frame speed" makes it think that human being's eyes are deceived and continuous action is seen, thereby, is that a main unit 52 changes a picture little by little for every frame, and can generate the animation effect on the television screen 60. In order to maintain the frame speed of this television, there is no co-processor 200, if it is ** in ***** about the new picture of per second 1/30 or 1/60. A co-processor 200 can also generate a series of continuation sound with the animation effect on the television screen 60.

[0028] [Operation of whole system] drawing 6 shows operation by the whole video game system 50 more to the detail, and since graphics are generated, drawing 7 shows all the steps performed by the video game system 50. In this example, a main processor 100 reads the video game program 108 memorized by main memory 300 (generally, this video game program 108 is stored in the video game storage 54 from the first, and is copied to main memory 300 from this video game storage 54).

Answering executing the video game program 108, (reaching and answering an input from the game controller 56) a main processor 100 is ***** (or it reads from a color television receiver 58) (drawing 7, block 120a) about the list 110 of commands to a co-processor 200. Generally this list 110 contains two kinds of commands.

(1) A graphics command (2) audio command graphics command directs what picture should be generated on the television screen 60 to a co-processor 200. An audio command directs what sound should be generated to a co-processor 200, in order to reproduce by the television loudspeaker 62.

[0029] Since a co-processor 200 controls what is displayed on the television screen 60, the list of graphics commands is called "display list." Since the list of audio commands controls the sound reproduced by the loudspeaker 62, it is called

"reproduction list." Generally, a main processor 100 specifies a new display list and a new reproduction list to each video "frame" time of a color television receiver 58.

[0030] In this example, a main processor 100 supplies display/reproduction list 110 to a co-processor 200 by memorizing the above-mentioned list to main memory 300, and next telling a co-processor 200 about the storage location (drawing 7, block 120c). A main processor 100 checks that main memory 300 is equipped with the graphic which stores all the data needed since the graphics and audio as which the co-processor 200 was required by display/reproduction list 110 are generated again, and the audio database 112. The thing or all with a graphic and the audio database 112 can be taken out from the video game storage 54. Display/reproduction list 110 specifies whether a co-processor 200 should use the portions of a graphic and audio database 112 throat. A main processor 100 can respond also to checking that the signal processor 400 has loaded "the microcode (computer program which directs what should be made to a signal processor by getting it blocked)."

[0031] A signal processor 400 reads display/reproduction list 110 in a main processor 100 (drawing 7, block 122a), and processes this list (drawing 7 (another data in a database 112 are accessed if needed), block 122b). A signal processor 400 generates the audio output data 114 for storing temporarily in the graphics viewing command 112 for the two main outputs, i.e., the further processing by the display processor 500, (drawing 7 and block 122c), and main memory 300. A signal processor 400 processes audio data in time very shorter than the time required in order to reproduce an audio by the loudspeaker 62. Another portion of the co-processor 200 called "audio interface" (not shown) reads succeeding the audio data by which the buffer was carried out, and outputs the audio data on real time for reproduction by the television loudspeaker 62.

[0032] A signal processor 400 can supply the graphical display command 112 to the direct presentation processor 500 through the internal path of a co-processor 200, or writes these graphical display commands in main memory 300 for reference by the display processor (not shown). These graphical display commands 112 order what the geometrical configuration specified with the specified property is drawn for ("a rendering is carried out") to the display processor 500 (drawing 7, block 126a). For example, as the line, the triangle, or the rectangle (polygon) could be drawn based on these graphical display commands 112 and all were specified by the graphical display command 112, the display processor 500 can apply a triangle and a rectangle by the specific color, and/or can stick a texture 116 (for example, leaves or the picture of the brick of a brick wall). A main processor 100 memorizes the texture picture 116 to

main memory 300 for access by the display processor 500. In order to instruct directly to the display processor 500, a main processor 100 can also write the graphical display command 112 for reference by the display processor 500 in main memory 300 directly.

[0033] The display processor 500 generates the digital display of a picture which should appear on the television screen 60 as the output (drawing 7, block 126b). This digital image is often called "bit map", and is memorized in the frame buffer 118 which exists in main memory 300. The display processor 500 is memorized using frame buffer 118b in main memory 300, in order to memorize the depth information on a picture. Another portion of the co-processor 200 called "video interface" (not shown) reads a frame buffer 118, and changes it into the video signal which supplies the content to a color television receiver 58 (drawing 7, block 127). Generally, the "double buffer" of the frame buffer 118 is carried out. A co-processor 200 writes the "following" picture in the half of a frame buffer 118, and, on the other hand, a double buffer means that a video interface reads other halves of a frame buffer 118.

[0034] The various steps mentioned above by drawing 7 show that "pipeline" processing is carried out in this example. A "pipeline" means that different operation is performed in the stage where graphics generating processings differ simultaneously. an easy example is the method of washing by many persons Washing in non-pipeline mode means doing all the related work (the iron to wash and to dry being covered, and being folded up and put away) for carrying out 1 lump's washing, before beginning the next lump's washing. In order to save time, those who do much washing do "pipeline" processing of the process of wash by folding up and performing simultaneously operation over which the iron to wash, and to dry is covered of tidying up, to two or more lumps' washing.

[0035] Similarly, in this example, "pipeline" processing of the operation performed with a main processor 100, a signal processor 400, the display processor 500, and the video interface 210 is carried out. For example, in this example, a main processor 100 can assemble a display list on two video frames beforehand, and, on the other hand, a signal processor 400 and the display processor 500 can process data about one video frame beforehand, and the video interface 210 processes the data about the on-going present video frame. Pipeline processing is carried out in order that the detailed graphics rendering step performed by the display processor 500 by block 126a may also make a speed performance the maximum so that it may explain below.

[0036] System-architecture] drawing 8 more detailed than [shows the more detailed architecture of the video game system 50. This drawing shows the video game main

unit 52. In addition to a main processor 100, a co-processor 200, and main memory 300, the video game main unit 52 is equipped with the element of an addition of the clock generation machine 136, the serial peripheral interface adapter 138, the audio digital to analog converter (DAC) 140, the audio amplifier/mixer 142, the video digital to analog converter 144, and video encoder 146 grade.

[0037] In this example, the clock generation machine 136 (this is controlled by the quartz resonator 148) generates the timing signal which takes the timing of other elements of a main unit 52, and a synchronization. A different main unit element requires a different clock frequency, and the clock generation machine 136 gives a suitable clock frequency output (or frequency from which a suitable clock frequency is obtained by carrying out dividing etc.). The block 216 in a co-processor 200 receives a clock signal from the clock generation machine 136, and distributes those clock signals to other circuits of each in a co-processor 200 (if required, after carrying out dividing suitably).

[0038] In this example, the game controller 58 is instead connected to the main unit 52 through the serial peripheral interface adapter 138, although it does not connect with the main processor 100 directly. The serial peripheral interface adapter 138 demultiplexes the serial data signal inputted from the game controller 56 (or other serial peripheral devices) to four sets (or five sets), is a predetermined format and gives this data to a main processor 100 through a co-processor 200. In this example, the serial peripheral interface adapter 138 is bidirection, i.e., the serial peripheral interface adapter 138 can transmit the serial information specified by the main processor 100 while receiving serial information.

[0039] In this example, the serial peripheral interface adapter 138 also has the "boot ROM" read-only memory 150 which memorizes a little initial program load (IPL) code. This IPL code memorized in "boot ROM" 150 is performed by the main processor 100 at the time of a start and/or reset, in order that a main processor 100 may enable it to start execution of initial game program instruction 108a in the video game storage 54 (refer to drawing 9, block 160a, and 160b). Next initial game program instruction 108a controls a main processor 100, initializes a driver and a controller required in order to access main memory 300 (refer to drawing 9, block 160c, and 160d), and in order to perform by the main processor 100 and the co-processor 200 and to use it, it copies a video game program and data to the more nearly high-speed main memory 300 (refer to drawing 9, Blocks 160e and 160f, and 160g).

[0040] Moreover, the serial peripheral interface adapter 138 is equipped with the security processor (for example, another small microprocessor) which communicates

with the security processor 152 (for example, small microprocessor) of the relation in the video game storage 54 in this example (refer to drawing 8). The pair (one side is in the video game storage 54, and another side is in a main unit 52) of this security processor 152 performs the authentication function to guarantee that only the permitted storage can use it with the video game main unit 52. Refer to U.S. Pat. No. 4,799,635. In addition to performing a security function under control of software, in this example, the security processor 152 in the serial peripheral interface adapter 138 processes the data which received from the game controller 56 under control of software.

[0041] Drawing 8 shows the connector 154 in the video game main unit 52. This connector 154 is connected with electric contact 74 of the edge of the printed circuit board 70 of the video game storage 54 in this example (refer to drawing 1). Thus, a connector 154 connects a co-processor 200 to storage ROM 76 electrically.

Furthermore, a connector 154 connects the storage security processor 152 of storage to the serial peripheral interface adapter 138 of a main unit. In the specific example, although a connector 154 is mainly written in and data and an instruction are read in the impossible read-only memory 76, a main unit sends information to the video game storage 54, and the main unit 52 is designed so that information can be read in the video game storage 54 so that a connector may be bidirection that is,.

[0042] Drawing 8 shows that it is processed by the electronic circuitry of the co-processor 200 exterior, before the audio and video outlet of a co-processor 200 are sent to a color television receiver 58. In this example, although especially the co-processor 200 outputs an audio and a video outlet by the digital format, generally the conventional home color television receiver 58 needs an analog audio and a video outlet. Therefore, the digital output of a co-processor 200 is changed into analog form. DAC140 performs about audio information and this function is performed by VDAC144 about video information. It is generated in the exterior of a main unit 52, and the analog audio output of DAC140 is amplified by the audio amplifier 142 to which mixture also carries out the audio signal supplied through the connector 154. The analog video outlet of VDAC144 is supplied to the video encoder 146. The video encoder 146 changes for example, the "RGB" input signal into a compound video outlet. The stereo audio output and the compound video outlet of the video encoder 146 by which the audio amplifier 142 was amplified are given to the home color television receiver 58 through a connector (not shown).

[0043] As shown in drawing 8, main memory 300 memorizes a video game program in the form of CPU instruction 108b. Such CPU instruction 108b is usually copied from

the video game storage 54. Although CPU100 of this example can carry out direct execution of the instruction from storage ROM 76, it is very larger than the time needed for accessing each instruction from main memory 300. [of the time needed for accessing each instruction from ROM] Therefore, a main processor 100 usually copies the game program / data 108a from ROM76 to main memory 300 according to the need for a block, and in order to actually execute an instruction, it accesses main memory (refer to drawing 9, block 160e, and 160f). Preferably, the main processor 100 has the internal cache memory, in order to decrease the instruction access time further.

[0044] Drawing 8 shows that the database of the graphic needed in order that the video game storage 54 may supply the specific graphic and specific sound of video game, and sound data 112a is also memorized. A main processor 100 reads graphics and sound data 112a in the video game storage 54 if needed, is the form of the texture data 116, sound data 112b, and graphics data 112c, and memorizes these data to main memory 300. The display processor 500 is equipped with the internal texture memory 502 by which the texture data 116 are copied to the interior if needed in this example.

[0045] The co-processor microcode 156 also memorizes the video game storage 54. As mentioned above, in this example, a signal processor 400 performs a computer program, in order to perform various graphics and audio functions. It is supplied by the video game storage 54, this computer program, i.e., a "microcode." microcode ***** from which different storage differs since a microcode 156 is given by the video game storage 54 -- the specific function which things are made and is supplied by the co-processor 200 under control of software by this -- ***** Generally, when starting a signal processor, a main processor 100 copies a part of microcode 156 to main memory 300 always, and a signal processor 400 accesses other portions of a microcode if needed. A signal processor 400 performs the microcode of the instruction memory 402 in a signal processor 400. Since the SP microcode 156 is too large to inputting all into the internal instruction memory 402 of a signal processor at once, in order that a signal processor 400 may enable it to perform another task, to load the portion of a different microcode to the interior instruction memory 402 of main memory 300 shell is needed. For example, a part of SP microcode 156 is loaded to a signal processor 400 for graphics operation, and, on the other hand, other portions of the SP microcode 156 are loaded to a signal processor for audio processing. In this example, the signal processor microcode RAM 402 (and additional signal processor data memory RAM which is not shown in drawing 8) is mapped by the address space of a main processor 100, consequently can carry out direct access of

the main processor to the content of RAM under control of software according to a load instruction and a write-in instruction.

[0046] [Main processor 100] The main processor 100 in this example is MIPS of California and Mountain View (Mountain View). Technologies, MIPS designed by Inc. R4300 RISC It is a microprocessor. This R4300 processor is equipped with an integer and the 64-bit register file for floating point arithmetics, 16KB of instruction cache, 8KB of write back data cache, and the EU that has the 32 entry TLB of the virtual address / physical address calculating. A main processor 100 executes the CPU instruction 108 by the kernel mode by 32 bit addresses. Although 64-bit integer operation can be used in this mode, for making a performance into the maximum, its 32-bit call rule is desirable. Please refer to Heinrich and MIPS microprocessor R4000 user's manual (MIPS Technologies, Inc, 1994, the 2nd edition) about the information beyond this about a main processor 100, for example.

[0047] A main processor 100 communicates with a co-processor 200 through a bus 102. The bus 102 is constituted from the 32 bit SysAD multiplex address of bidirection / data bus, the 5 bit wide SysCMD bus of bidirection, the further control line, and the timing line by this example. Please refer to Chapter 12 and it of the above-mentioned Heinrich manual or subsequent ones.

[0048] R4300 conventional main processor -- six hardware interruption and the one interior (timer) -- it interrupts and two software interrupts and one non-mask interruption (NMI) are supported In this example, three inputs of these six hardware interruption inputs (INT0, INT1, and INT2) and the input of this one non-mask interruption (NMI) enable other portions of the video game system 50 to interrupt a main processor. It connects in order for interruption INT0 of a main processor to enable a co-processor 200 to interrupt a main processor in detail, it connects in order for interruption INT1 of a main processor to enable video game storage 54 to interrupt a main processor, and the interruption INT2 and NMI of a main processor is connected in order to enable the serial peripheral interface adapter 138 to interrupt a main processor 100. When interrupting, a main processor always answers this (reading a status register or performing other suitable processings for example) by the suitable method with reference to an internal interruption register next, in order to determine the cause of the interruption. The NMI interruption input from almost all the serial peripheral interface adapters 138 can carry out a mask (it is got blocked and those inputs are alternatively made as for a main processor 100 to an enable and a disable under control of software).

[0049] Through the CPU-co-processor bus 102, a main processor 100 reads data in

the remaining portion of the video game system 50, and writes in data there. A co-processor 200 performs a memory-mapping function, and can be made to carry out the address of the main processor 100 to "boot ROM" 150 in main memory 300, the storage cartridge ROM 76, and the serial peripheral interface adapter 138 (and other portions of the serial peripheral interface adapter 138), each portion (for signal processor RAM402 to be included) of a co-processor 200, and other portions of the video game system 50.

[0050] In this example, operation performed by the main processor 100 is completely dependent on the video game program 108. In this example, all "system" software is supplied from storage 58, in order to give the greatest flexibility. Different video game (or other applications) is the high-level software of a different kind, and can be performed more efficiently. Therefore, the main unit 52 in this example does not offer any standard software libraries (or all software libraries). This software library is because flexibility may be restricted. Instead, in this example, all software is supplied by the video game storage 54.

[0051] In order that the developer of the video game program 108 may manage various kinds of resources in the video game system 50, it may be thought that he wants to use new style sorting wear architecture, for example, a device driver, a scheduler, and a thread library. Since main processors 100 are the latest RISC processor / computer, they are suitable for executing using such a software architecture/composition and the video game program 108 in a high-level software environment.

[0052] The example of the system "a memory map" of the address space of a main processor 100 is shown in drawing 10. Main memory 300 is divided into two banks (a bank 0 and bank 1) in this example as shown in drawing 10. Furthermore, the register 307 of the specific composition in main memory 300 is mapped by the address space of a main processor 100 like the register in a co-processor 200. A main processor 100 can control each of various co-processor subblocks by this example by writing in the control register relevant to each subblock of a co-processor 200 under control of the video game program 108.

[0053] The address space of the video game storage 54 is divided into two (as opposed to two different devices) "fields" as shown in drawing 10. These "fields" is mapped by some portions of the address space of a main processor 100. Each portion (getting it blocked the PIF boot ROM 150, the PIF buffer RAM, and a PIF status register) of the serial peripheral interface adapter 138 is also mapped by the address space of a main processor 100.

[0054] [Integrated main memory 300] The main memory 300 of this example is

Rambus of California and Mountain View. It consists of the RDRAM dynamic RAM which can come to hand from Inc. In this example, although main memory 300 is extensible so that the storage capacity to 8 megabytes may be given, in order to lower cost, it should ship the main unit 52 by RAM (for example, 2 or 3MB) of small capacity. [0055] Main memory 300 gives storage capacity about the video game system 50 whole in this example. This offers the single address space (refer to drawing 10) which memorizes all important data structures including the following data structures (it was shown in drawing 8 like).

– Data which communicated between each part of main processor instruction 108, signal processor microcode 156, display list graphic command 110a, reproduction list audio command 110b, the texture map 116, and the working value and system of the working value and co-processor of other graphical data 112c, color picture frame buffer 118a, (depth Z) buffer 118b, sound data 112b, audio output-buffer 114, and main processors [0056] The advantage and fault which use single address space memory architecture for a raster scan display system are known (see the "foundation [of computer graphics], and practice" 177–178 page (2nd edition Addison–Wesley 1990) besides Foley). The architecture of a video game (and other graphics) system of the former many used the exclusive Video RAM device for graphics data, and used the memory device of other form for the data of other form, and had refused the architecture of a single address space. However, the integrated main memory 300 gives many advantages in this specific example of the video game system 50. For example, it is as follows.

[0057] The data communication between system elements is simplified. the overhead of the addition at the time of communicating other portions and data of a system, once data are memorized by main memory 300 -- a few -- the need -- or it is not required at all The overhead which transmits data between the portions from which a system differs is made into the minimum. For example, since each subblock in a main processor 100 and a co-processor 200 can access the main memory 300 of a system, respectively, the main memory 300 used with all system elements in order to memorize a data structure can be used also as the general-purpose communication channel / a data buffer between elements.

[0058] For example, direct access of the display list 110 which the main processor 100 memorized in main memory 300 can be carried out by the signal processor 400. Similarly, direct access of the viewing command which the main processor (and/or, signal processor) memorized in main memory can be carried out by the display processor 500. the working data (automatically written in main memory 300 through a

"cache flash plate") of a main processor 100 -- a system -- all -- others -- it can use also for a portion immediately

[0059] Integrated memory gives the flexibility of memory allocation. It can be used, in order for the location of main memory 300 to seem to be alike, therefore for each location to memorize the data structure of what form. The determination of all allocation of main memory 300 is left to the programmer of application. This is the size of a data structure, and the point of use of memory, and gives big flexibility. A data structure can be memorized at the arbitrary places in main memory 300, and each location of main memory 300 can be assigned even to how in the place which the programmer of application pinpointed.

[0060] For example, a certain video game programmer can determine to use a smaller frame buffer so that the large frame buffer for a high resolution picture and/or picture scrolling, and panning may be offered and other video game programmers may release room to other data structures (for example, a texture or audio data) on the other hand. a certain application -- many of storage capacity of main memory 300 -- an audio data structure -- and a few can be turned to graphical data and, on the other hand, other applications can turn many of storage capacity to graphic related data the same video game program 108 can shift memory assignment from a portion with a game play to other portions at a dynamic, in order to realize a different effect (for example, when a game changes level) The flexibility of application is not restricted by the ball hard-wired memory allocation fixed up.

[0061] Integrated RAM architecture supports the share ring of a flexible data structure, and use. Since all important data structures are memorized in the common main memory 300, they can all be accessed with a main processor 100 and an alien-system element. There is no distinction of hardware between a display image and a source picture. For example, if a main processor 100 is required, it can carry out direct access of each pixel in a frame buffer 118. The scanning conversion output of the display processor 500 can be used as a texture for texture mapping processing. A picture source data and the image data by which scanning conversion was carried out can exchange in order to realize carrying out the warping of the picture by which scanning conversion was carried out in the special effect, for example to a view, and/or it can combine.

[0062] The fault of integrated memory architecture (for example, competition for access to the main memory 300 by the portion from which a system differs) was made into the minimum by the careful system design. Although main memory 300 is accessed through the single narrow (9-bit width of face) bus 106 in this example,

permission bandwidth is realized by making a bus very high-speed (to for example, 240MHz order). The video game system 50 whole is equipped with the data cache so that a margin to wait for main memory 300 to become usable may be given to each sub element more.

[0063] [Co-processor 200] drawing 8 shows that the co-processor 200 is equipped with some elements in addition to a signal processor 400 and the display processor 500. That is, they are the following elements:

- CPU interface 202 and serial interface 204- parallel — the DRAM controller / interface 212, main interface bus 214, and the timing block 216 of peripheral-interface-adapter 206, audio interface 208, video interface 210, and main memory — in this example, the main bus 214 makes possible the thing of the main various elements in a co-processor 200 communicated mutually, respectively

[0064] It is shown that drawing 11 is more detailed drawing of a co-processor 200, and is the meeting of the processor that a co-processor 200 operates in parallel [all become active simultaneously and], a memory interface, and a control logic. The following explains briefly all the functions of other subblocks of these of a co-processor 200 that are alike, respectively and are offered more.

- A signal processor 400 is a microcode-ized engine which performs the task of an audio and graphics.
- The display processor 500 is a graphics display pipeline who does a rendering to a frame buffer 118.
- The co-processor serial interface 204 gives the interface between the serial peripheral interface adapter 128 and a co-processor 200 in this example.
- The co-processor parallel peripheral interface adapter 206 interfaces with the video game storage 54 or other parallel devices which were connected to the connector 154.
- The audio interface 208 reads information in the audio buffer 114 in main memory 300, and outputs it to an audio DAC 140.
- The co-processor video interface 210 reads information in frame buffer 118a in main memory 300, and outputs it to video DAC 144.
- The CPU interface 202 is the gate between the remaining portions of a main processor 100, a co-processor 200, and the video game system 50.
- A DRAM controller / interface 212 is paths where a co-processor 200 (and main processor 100) accesses main memory 300 through this. The memory interface 212 gives access to main memory 300 to a main processor 100, a signal processor 400, the display processor 500, the video interface 210, the audio interface 208, a serial, and parallel interfaces 204 and 206.

Various processors and each of an interface become active simultaneously.

[0065] In this example, the signal processor 400 has the instruction memory 402, data memory 404, the above-mentioned Scala processing unit 410, and the above-mentioned vector-processing unit 420. The instruction memory 402 has memorized the microcode for execution by the Scala processing unit 410 and/or the vector-processing unit 420. Data memory 404 has memorized the input data, the work data, and the output data to the Scala processing unit 410 and the vector-processing unit 420. In this example, although a signal processor 400 executes only the instruction from the instruction memory 402, it is accessed to main memory 300 with a direct-memory-access (DMA) method.

[0066] At this example, the Scala processing unit 410 is MIPS. It is the general-purpose integer processor which performs the subset of the instruction set of R4000. This is used in order to do the general-purpose work specified by the microcode in the instruction memory 402. The vector-processing unit 420 consists of eight 16-bit calculation elements which can perform numerical calculation in parallel. Especially the vector-processing unit 420 is suitable for graphics matrix calculation and digital audio signal processing work of a certain kind.

[0067] The display processor 500 is the graphics display pipeline engine who makes the digital display of a display image perform in this example. This operates based on the graphics viewing command generated by the signal processor 400 and/or the main processor 100. In addition to the texture memory 502, the display processor 500 is equipped with a rasterizer 504, the texture unit 506, the color combiner 508, the blender 510, and the memory interface 512. Simply, a rasterizer 504 rasterizes a polygonal (for example, a triangle and a rectangle) geometric primitive, in order to determine, which pixel on the display screen 60 is within the limits of these primitives, and. The texture unit 506 sticks the texture map memorized in the texture memory 502 on the texture field decided by the primitive edge equation solved by the rasterizer 504. The color combiner 508 combines the color of a texture, and the color relevant to a graphic primitive, and interpolates the meantime. A blender 510 blends the obtained pixel and the pixel in a frame buffer 118 (the pixel in a frame buffer is accessed through the memory interface 512), and concentrates also on performing Z buffering (getting it blocked hidden surface removal and anti-aliasing operation). It reads, and correction and write-in operation are performed, and texture memory 502 is loaded / copied, a rectangle is corrected (high-speed clearance), and the memory interface 512 has the special mode to each pixel in which a multiplex pixel is copied to a frame buffer 118 from the texture memory 502. The memory interface 512 has one

or more pixel caches, in order to reduce the number of times of access to main memory 300.

[0068] The display processor 500 has the circuit 514 which memorizes the state of a display processor. This status information is used by the remaining portion of the display processor 500 in order that the rendering of all precedence that chooses for example, rendering mode and is performed by mode change may ensure generating before mode change is performed.

[0069] The command list for display processor 500 usually carries out direct ingress of the signal processor from a signal processor 400 through exclusive "X bus" 218 linked to a display processor. In detail, in this example, the X bus 218 is used in order to transmit a graphics viewing command to the command buffer in the display processor 500 (not shown to drawing 11) from the signal processor memory 404 for processing by the display processor 500. However, in this example, a signal processor 400 and/or a main processor 100 can also supply a graphics viewing command to the display processor 500 through main memory 300.

[0070] The display processor 500 is loaded to the internal texture memory 502, reads a frame buffer 118 for a blend, reads Z-buffer 118B for depth comparison, writes it in Z-buffer and a frame buffer, and in order to read the arbitrary graphics viewing commands memorized by main memory, a physical address is used for it and it accesses main memory 300.

[0071] [Interior bus architecture of co-processor] drawing 12 is detailed drawing rather than the example of the composition of the co-processor bus 214 is shown. This composition consists of 32 bit-address ("C") bus 214C and 64-bit data ("D") bus 214D in this example. These buses 214C and 214D are connected to each of a signal processor 400, the display processor 500, the CPU interface 202, the audio interface 208, the video interface 210, a serial interface 204, the parallel peripheral interface adapter 206, and the main memory (RAM) interface 212. As shown in drawing 12, each subblock of a main processor 100 and a co-processor 200 communicates with main memory 300 through the internal co-processor buses 214C and 214D, and the main memory interface / controllers 212a/212b.

[0072] In this example, in order to communicate through the 9-bit wide main memory multiplex address / data bus 106, a main memory interface / controllers 212a/212b change into a 9-bit wide format the main memory address which appeared on co-processor address bus 214C, and perform conversion with the 9-bit wide data format of the main memory bus 106, and the 64-bit wide data format of co-processor data bus 214D. At this example, a DRAM controller / interface 212 is Rambus as the

part. It has the conventional RAM controller 212b (refer to drawing 12) offered by Inc. By use of the main memory bus 106 of 9-bit width of face, the number of chip pins of a co-processor 200 is reduced.

[0073] In this example, the subblock of illustration of each co-processor 200 has the direct-memory-access (DMA) circuit of relation which makes it possible to carry out the address to main memory 300, and to access it uniquely. For example The single processor DMA circuit 454, Each of the display processor DMA circuit 518, the audio interface DMA circuit 1200, the video interface DMA circuit 900, the serial interface DMA circuit 1300, and the parallel peripheral-interface-adaptor DMA circuit 1400 The co-processor subblock of relation generates the address on co-processor address bus 214C. And it makes it possible to communicate data through co-processor address bus 214D (in order that the display processor 500 may access a frame buffer 118 and the texture data 116 further). It has another memory interface block 512.

[0074] Although the subblock of each co-processor 200 can access main memory 300 uniquely, in this example, they are sharing common buses 214C and 214D. Moreover, only one of the subblocks can use those common buses at once. Therefore, the co-processor 200 is designed so that those common buses 214 may be used most efficiently. For example, the subblock of a co-processor 200 can carry out a buffer or the "cache" of the information so that different bus access frequency by the same subblock may be made into the minimum and a subblock may have a margin more to temporary bus use impotentia. A signal processor 400 enables it, as for a dedicated bus 218, to communicate that the main bus 214 becomes usable with the display processor 500, without waiting.

[0075] As shown in drawing 12, each subblock of a co-processor 200 contains the control/status register which can be accessed by the main processor 100 through the CPU interface 202. For example, the signal processor register 407, the display processor register 507, the audio interface register 1207, the video interface register 907, the serial interface register 1307, the parallel peripheral-interface-adaptor register 206, RAM interface register 1007a, and RAM controller register 1007b are mapped by the address space of a main processor 100, respectively. Then, in order to control directly operation of the subblock in a co-processor 200, under control of the video game program 108, a main processor 100 can read these various registers, and/or can write them in.

[0076] [Signal processor 400] drawing 13 shows the architecture of the signal processor 400 of this example in more detail. As mentioned above, the signal processor 400 is equipped with the Scala unit 410, the vector unit 420, the instruction

memory 402, and data memory 404. In this example, the Scala unit 410 is 32 binary-integer processor which performs the subset of MIPS4000 instruction set (it defines as the bottom of the architecture of MIPS4000 as a "CP1" co-processor of the Scala unit 410). The vector unit 420 performs integer calculation (for example, multiplication, addition, subtraction, and composite/accumulation) in parallel about the value of eight bits sets [16].

[0077] The vector unit 420 can perform the same operation simultaneously in parallel about eight pairs of 16-bit operands. Thereby, a signal processor 400 comes to suit a "sum of product" operation, for example, matrix multiplication, texture re-sampling, audio digital signal processing, for example, digital audio composition, and space / frequency filtering especially.

[0078] The signal processor 400 is using RISC (RIDEYUSU instruction set computer) architecture, in order to perform highly efficient machine control based on the instruction which exists in the instruction memory 402. The EU is equipped with the program counter 432 used in order to carry out the address to the instruction memory 402 through a path 434 in this example. Although to arrange first all instructions that should be executed by the signal processor 400 in the instruction memory 402 is needed in this example, a program counter 432 can access only the 4-K byte instruction space in the instruction memory 402. EU 430 generates the output-control signal 436 based on the specific instruction executed now. The output-control signal 436 controls other all of signal processors 400, and it is set in order in order to manage pipeline instruction processing. The Scala unit 410 and the vector unit 420 are controlled by these control signals 436. For example, the address of the Scala unit 410 is carried out to data memory 404 through a path 438, and it reads data in data memory 404 using loading / storage block 440, and/or writes data there. A data path 414 tests according to the result of an operation, and gives the obtained status output to EU 430 through a path 442. EU 430 uses these status outputs, performs a conditional branch or a jump, and loads the program counter 432 which has the suitable address (degree) to the instruction memory 402. Since the Scala unit 410 is equipped with such broader capacity, in addition to a 32-bit integer arithmetic, the Scala unit 410 is used for general-purpose functions, such as flows of control and address computation, in this example.

[0079] EU 430 is Standard MIPS. According to R4000 instruction set, middle, a jump, and a register instruction format are performed. Drawing 14 (a) shows how one example of the register instruction format 450 is shown, and a signal processor 400 accesses three 128-bit wide WORD 452 in data memory 404 using this register

instruction format. The register instruction format 450 has the 6-bit operation-code field 450(a) 5-bit source register regulation child 450 (b) and 5-bit target (source/destination) register regulation child 450(c) 5-bit destination register regulation child 450 (d) and the parameter field 450 (e). A parameter field 450 (e) can specify a shift amount and/or a function, and the operation-code field 450 (a) defines further the operation which should be performed. Each of the field 450 (b), 450 (c), and 450 (d) specifies the location in data memory 404, and each specifies a 128-bit word. [0080] The vector unit 420 treats each of a 128-bit word as a connection sequence of eight 16-bit values, and calculates each of a 16-bit value in parallel as shown in drawing 14 (b). The operation of the vector unit 420 is usually MIPS. It is called by the instruction in CP1 type instruction stored in R4000 instruction set for the floating point arithmetic (in this example, the signal processor 400 does not have the floating point unit).

[0081] The Scala unit 410 has the register file 412 which builds in 32 registers. Each register is 32-bit width of face. The Scala unit 410 has the data path 414 which includes an integer arithmetic and other operations for an adder circuit required for an execution sake, a shift circuit, and other logical circuits. A register file 412 is MIPS. It is the same as that of the general-purpose register file defined by R4000 architecture, and the instruction in R4000 format is received. A data path 414 is equipped with an integer multiplier / divider, and operates with EU 430 which receives a 64-bit width-of-face instruction from the instruction memory 402.

[0082] the vector unit 420 -- 8 sets of register file 422 (0) -- it has 422 (7) and 8 sets of correspondence data path 423(0) --423(7) The data path 423 has the 16-bit multiplier, the 16-bit adder, and the 48-bit accumulator, respectively (48-bit accumulation also suits the partial product for which a series of 16-bit multipliers and sums are used, in order to obtain a 32-bit result to the graphics operation which an audio filter is fitted to many taps, and requires beyond 16-bit precision). Each of a register file 422 has the register of 32-bit width of face of 32 pieces. The data path 444 of 128-bit width of face connects the vector unit 420 to loading / storage block 440, and the data path 446 of another 128-bit width of face connects loading / storage block 440 to data memory 404. Data memory 404 memorizes 4096 (4KB) words, and each word is 128-bit width of face. It is sliced by eight 16-bit segments when the word in data memory 404 is searched for use by the vector unit 420. Each segment is sent to another register file 422 in the vector unit 420 (refer to drawing 14 (b)). Drawing 14 (c) shows the example of the add operation performed by the vector unit 420. When the vector unit 420 writes in the destination by which addressing was

carried out within data memory 404, before [a register file 422] being written in data memory, it becomes 16 bits combined with a 128-bit word, respectively (refer to drawing 14 (a)). Moreover, loading / storage block 440 is equipped with ***** steering multiplexer structure (not shown) for 16-bit sub language from another register file 422 or another register file 422 among the 128-bit words in data memory. Under the present circumstances, specific sub language and a specific specific vector unit register file can be chosen based on the instruction from the instruction memory 402. the data unit (for example, a byte, 16-bit half language, or a 32-bit word) of the size from which similarly loading / storage block 440 differs between data memory 408 and the Scala unit 410 -- ***** -- it has another steering multiplexer structure (not shown) Under the present circumstances, a specific data unit and specific size can be specified by the instruction in the instruction memory 402. For example, Heinrich, MIPS Refer to loading of the "byte" in R4000 microprocessor user's manual (2nd edition . 1994), a "halfword", "WORD", the "WORD left", and a "WORD right", and the publication of storage.

[0083] The signal processor 400 has DMA controller 454 and the CPU control register 456. In order to connect with the interior bus 214 of a co-processor and to transmit data to the instruction memory 402 and/or data memory 404, or since data are transmitted from the instruction memory 402 and/or data memory 404, DMA controller 454 is used. For example, DMA controller 454 can copy the microcode module 156 to the signal processor instruction memory 402 from main memory 300. DMA controller 454 is used in order to transmit information between data memory 404 and main memory 300 again. It is ordered by EU 430 and DMA controller 454 receives the DMA address and data information from the Scala unit data path 414 through a path 438. DMA controller 454 is ordered by the main processor 100 through the CPU control register 456 again. The CPU control register 456 is mapped by the address space of a main processor 100, and is accessed by a signal processor 400 and EU 430 using a MIPS "CP0" instruction format.

[0084] Drawing 15 (d) - drawing 16 (i) show the example of the CPU control register 756. The register shown in drawing 15 (d) - drawing 15 (h) is used in order to control and/or supervise DMA controller 454.

[0085] for example, SP-DRAM shown in drawing 15 (d) Since it is written in by the main processor 100 (SP EU 430 -- the same), or is read and the start DMA address in the instruction memory 402 or data memory 404 is specified, the DMA address register 458 is used. Since the start DMA address in main memory 300 is specified, the SP memory DMA address 460 shown in drawing 15 (e) is used. Reading and the

write-in DMA length registers 462 and 464 which were shown in drawing 15 (f) and drawing 15 (g) specify the length of the data block which should be transmitted between a signal processor 400 and main memory 300, respectively. Which one side of these two registers since the block length is specified, should be used responds, and the direction of a transfer is decided. The DMA status registers 466 and 468 shown in drawing 15 (h) – drawing 15 (i) are read by the main processor 100 in order to determine whether DMA controller 454 is full, i.e., a busy, respectively.

[0086] Drawing 16 (j) shows the main SP status register 470 in the CPU control register 456. The SP status register 470 acts as an SP control register, when written in by the main processor 100 (above figure in drawing 16 (j)), and when read by the main processor 100, it directs SP state (the following figure in drawing 16 (j)). When used as a status register, the SP status register 470 is. (field 472) [whether SP is stopped or (field 471) SP is operating in breakpoint mode, and] [whether DMA controller 454 is a busy (field 474) or full (field 475) and] (field 477) [whether SPI/O is full or (field 476) SP is operating by single step mode, and] In order that it may operate in the mode in which it does not interrupt when SP reaches a break point or (field 478) SP may give the state about various software dependence parameters A main processor 100 is told about whether various wide use "signal" 479 which the bottom of control of software can define were generated. A main processor 100 stops or starts a signal processor 400 by writing in a register 470 (fields 480 and 481). Break point mode can be cleared (field 482), interruption mode can be cleared or set (fields 483 and 484), single step mode can be cleared or set (fields 485 and 486), and interruption can be cleared or set in break point mode (fields 487 and 488), and various software dependence "a signal" can be cleared or set.

[0087] Drawing 16 (k) shows another SP register 491 used as a "semaphore" for the general-purpose communication between a main processor 100 and a signal processor 400. The SP register 491 was set when a main processor 100 read a register, and it is equipped with the flag cleared in case it writes in a register. A signal processor 400 can also set or clear this flag.

[0088] Drawing 16 (l) shows the BIST status register 492 of SP instruction memory. A BIST state is shown when the BIST status register 492 is used as a BIST control register when written in by the main processor 100 (above figure in drawing 16 (l)), and read by the main processor 100 (the following figure in drawing 16 (l)). A program counter 432 is preferably mapped by the CPU control register 456 again so that it may be written in by the main processor 100 and can be read.

[0089] The specific function which the [microcode of signal processor] signal

processor 400 performs is decided by the SP microcode 156 given by the video game storage 54. In this example, the SP microcode 156 gives both graphics and an audio processing facility. As mentioned above, the main task performed by the signal processor 400 to graphics operation includes reading of a display list, execution of 3D geometrical conversion and a lighting operation, and generating of the correspondence graphics viewing command used by the display processor 500. In a detail, a signal processor 400 performs all the following graphics functions under control of a microcode 156 more.

- The setup flow control signal processor 400 of a definition / loading clipping of a definition and peak generating of the processing and the matrix of a display list, and a lighting texture, and a curling and a display processor command performs all the following functions under control of a microcode 156, in order to process an audio.

- Writing of a digital audio sample to processing and digital audio composition / processing, and the main memory audio buffer 114 of a reproduction list [0090] The [task list] main processor 100 tells a signal processor 400 about what should be made by giving a task list to a signal processor. The program of the microcode 156 which runs on a signal processor 400 is called task. A main processor 100 (and video game program 108 supplied by the video game storage 54) can respond to carrying out scheduling of the task on a signal processor 400, and calling it. This task list includes all information required for a signal processor 400 to perform a task including the pointer about the routine of the microcode 156 required to run to the well which performs a task. A main processor 100 supplies this task list under control of the video game program 108.

[0091] Drawing 17 shows one example of a task list 250. Refer to 1 or one or more display lists, and/or the reproduction list 110 for this task list 250. Refer to the another data structure which next includes other display lists or a reproduction list for this display list and/or the reproduction list 110. A display list 110 can specify an another display list and/or another graphical data. Similarly, refer to another reproduction list and/or the sound data for a reproduction list. In this example, a display list and a reproduction list can be considered as a hierarchy data structure of the depth to 10 level. A signal processor 400 processes the display list of a stack, and a reproduction list, and pushes and carries out pop [of the pointer of the present display list]. All display lists are ended by " and " command. For example, refer to other display lists 110 (2) for the display list 110 (1) shown in drawing 17. Refer to the graphical data 112 needed in order to perform this list for a display list 110 (2). Similarly, refer to the sound data 112b for the reproduction list 110 (4) shown in

drawing 17.

[0092] It is desirable to carry out the "double buffer" only of the portion of the display list 110 which changes for every frame about graphics animation. Thus, it is required to carry out the "double buffer" only of the data which change to the degree of a frame, it does in this way and space can be secured in main memory 300. The swapping between double buffers is effectively performed changing the segment base address in a task list 250, and by organizing a hierarchization display list by the efficient method appropriately. The fragmentation of a display list or a display list can be connected with one because of still more efficient memory use.

[0093] Drawing 18 shows the example of the processing performed by the main processor 100, in order to call processing of a new task list by the signal processor 400. A main processor 100 loads a task (display) list in main memory 300 first (block 601). A main processor 100 stops a signal processor 400 by writing in the SP status register 470 next, and/or reading in there (block 602 (or it checks in order to guarantee that the signal processor was stopped)). Next, a main processor 100 is written in the SPDMA registers 458, 460, and 462, and loads an initial microcode module to the instruction memory 402 of a signal processor (block 604, drawing 18). Next, a main processor 100 memorizes the address in the main memory 300 of the task (display) list loaded by block 601 to the data memory 404 of a signal processor (block 606, drawing 18). A main processor 100 writes this in the SP status register 470, in order to reset the program counter 432 of a signal processor next (block 608, drawing 18) and to start a signal processor 400 (block 610, drawing 18). Next, a signal processor 400 uses DMA controller 454, in order to usually incorporate a task (display) list from main memory 300 to data memory 404.

[0094] A start of the signal processor 400 equipped with the task list advances execution of each operation demanded by the task list. A signal processor 400 waits to continue performing a task list until processing reaches the end of a task list, and to suspend processing in the time of an end, and for a main processor 100 to give a new task list. Generally, although a main processor 100 gives a new task list only at once about each video frame, as mentioned above, in many cases, the display and/or reproduction list of task lists which reach in part and/or a task list refers to actually change for every frame. The "double buffer" of the portion of the task list in main memory 300 is carried out, therefore a main processor 100 can be written in a certain buffer, and, on the other hand, can read a signal processor 400 in another buffer. Before the following video frame, a main processor 100 changes a pointer and gives access to a new buffer at a signal processor 400.

[0095] When a signal processor 400 performs a task list, SP microcode 156 another module is searched from main memory 300 if needed [of performing a specific task]. For example, a signal processor 400 uses the DMA facility 454, loads a specific graphic microcode to the instruction memory 402, and executes the graphic command specified by the task list. Similarly, a signal processor 400 searches and loads an audio processing microcode routine, and performs audio processing specified by the task list. It is loaded if needed, different microcode routine, i.e., "overlay", and it processes the graphics of a specific form, and/or audio processing operation still the more nearly optimal. As one example, a signal processor 400 loads a special lighting graphics routine as overlay, performs specific lighting operation, and loads a clipping routine, i.e., overlay, and performs specific curling operation. Since the instruction memory 402 of a signal processor is designed so that it may not have sufficient size to memorize all of the SP microcodes 156 and a signal processor 400 can execute only the instruction from the internal instruction memory, it is required for a signal processor 400 during one execution of a task list 250 loading and RIRODINGU [a microcode].

[0096] Drawing 19 shows the example of the simplified graphics operation performed by the signal processor 400 based on a display list 110. In this simplified processing, a signal processor 400 is first ordered to perform a display list 110, and it sets the various attributes which define the whole graphics picture in which a rendering should be carried out by the co-processor. These attributes contain shading, a lighting, Z buffering, texture generating, fogging, and curling (drawing 19, block 612). Below, a signal processor 400 is ordered to perform a display list 110, and it defines modeling / viewing matrix, and a projection matrix (drawing 19, block 614). If a matrix suitable at once is defined, a signal processor 400 will be ordered to perform a display list 110, and the set of the peak will be changed based on the attribute set by block 612 based on the modeling / viewing matrix, and the projection matrix which were defined by block 614 (drawing 19, block 616). Finally, a signal processor 400 is ordered to perform a display list 110, and the graphical display (for example, triangle) command it is directed to the display processor 500 that carries out the rendering of the primitive is generated based on the vertex generated by block 616, and the attribute set by block 612 (drawing 19, block 618). A signal processor 400 answers Step 618 and transmits the display processor command (address of the command which it was got blocked and the signal processor 400 memorized in data memory 404 or main memory 300) generated for access and execution by the display processor 500.

[0097] Drawing 20 shows the whole processing 620 performed by the graphic microcode 156 of a signal processor 400, in order to process a display list 110 (for

example, in order to perform processing of the form shown in drawing 19). A signal processor 400 receives the following display list command, and it determines it in what kind of command (drawing 20, block 622). Generally the display list command of this example has five different form.

– The attribute command, the display processor command matrix command, vertex command, and triangle command flow control command of a signal processor [0098]
As the signal processor 400 was specified by the command as a display list command is an attribute command of a signal processor, the attribute of a signal processor is set (drawing 20, block 624). SP attribute command of the following form is defined by this example.

– Below shading, a lighting, and Z buffering texture ring fogging curling are the examples of SP attribute command format and a definition of relation.

[0099] [The attribute command of a signal processor]

G SETGEOMETRYMODE: [Table 1]

This command "sets" a thing with a rendering pipeline state. This state is held within a signal processor 400, and a user is provided with a set / clear interface. The bit which is in "ON" in the command field is turned ON by the internal state.

G Make a SHADE vertex shading possible or make usable the primitive color which carries out the paint of the polygon (at the time of a default, it is a vertex shading).

G LIGHTING lighting calculation is enabled.

G Make possible SHADING SMOOTH smooth jamming flat shading (at the time of a default, it is flat shading by the clearance of this bit).

G ZBUFFERZ buffer depth calculation is enabled.

G Automatic generating of the TEXTURE GEN texture coordinates S and T is enabled. After conversion, in order that globular form mapping may transpose S and the T value which were given at the beginning to the peak, it is used.

G Make usable the fog coefficient by which FOG generating should be carried out, and replace peak alpha. If alpha is large, it will become FOGI to distance more.

G Make possible linearization of the texture coordinate generated when TEXTURE GEN LINEARG TEXTURE GEN is set. For example, thereby, a panorama texture map becomes usable at the time of execution of environmental mapping.

G Generating of a details level (LOD) value is enabled for possible and texture edge mode at the texture by which the LOD MIPPU map was carried out.

A G_CULL FRONT positive polygon is chosen.

G Choose a CULL BACK backward polygon.

[0100] G Although it is the same as CLEARGEOMETRYMODE:G

SETGEOMETRYMODE, this command "clears" some of some things of a rendering pipeline state (the bit which is in "ON" in the command field is turned OFF by the internal state).

[0101] G LIGHT: [Table 2]

This command sends light to a rendering pipeline. In addition to ambient light, there is a directive (a number was assigned with 1-7) light to seven. A parameter specifies the number (n) of the light which should be transposed to this optical description. How many should be used among eight light, and since it specifies, the G NUM LIGHTS command is used. If the number of the specified light N Becomes, N light (1-N) should be used and the N+1st light will be ambient light. The "parameter" field should be set based on the value currently held data memory $404+(n-1) \times 2$.

[0102] Ambient light should be defined by a color r, i.e., light, Light g, and Light b (sign-less 8 binary integers), and this should be set to the color of the ambient light multiplexed by the color for drawing (when the lighting of the object by which texture mapping was carried out is carried out, the color of ambient light is used). (As ambient light, Light x, Light y, and optical z field are disregarded.) In this example, ambient light cannot turn OFF other than specifying black.

[0103] A directive light should be defined by a color r, i.e., light, Light g, and Light b (sign-less 8 binary integers), and this should be set to the color of the light source multiplexed like ambient light by the color for drawing. A directive light also has a direction. Light x, Light y, and optical z field (8-bit decimal with a sign which has a 7-bit decimal) show the direction of [from the object to illuminate]. (when G LIGHTTNG is made usable by the G SETGEOMETRYMODE command) There is a directive light made at least one ON. However, when the color is black, there is no influence in a scene.

[0104] Even when the number of light is not changed, the G NUM LIGHTS command should always be used before the following G VTX command after the G<U>LIGHT command.

[0105] G NUM LIGHT: [Table 3]

This command determines how many light should be used. This should always be used before the following G VTX command after the G LIGHT command. This parameter specifies the number of seven or less or more at least 1 sources of the diffused light (N). The source of ambient light is N+1 light, and the directive light source is the light by which a number was assigned to N from 1.

[0106] G SETOTHERMODE H: [Table 4]

This command sets the high WORD in the others [" / in blending, a texture ring, and

the display processor 500 containing the parameter of a frame buffer]" mode. a signal processor -- 400 -- being easy -- a set command -- an interface -- being shown -- a sake -- a display -- a processor -- 500 -- inside -- " -- others -- " -- a state -- highness -- and -- a low -- WORD -- memorizing -- ***** . It is analyzed by the signal processor 400 and this command is interpreted, although it is the command of a display processor. Therefore, this command cannot perform probably sending to a display processor directly, without letting a signal processor pass.

[0107] Since the following masks are constituted, the shift in this command and a length parameter are used.

$(0x01 \ll \text{length}) (-1) (\ll \text{shift})$

This mask is used in order to clear these bits in the status word of the display processor 500. OR of the new bit from a WORD parameter is carried out to a status word (the preliminary shift of the parameter word is carried out).

[0108] G Although it is the same as SETOTHERMODE L:G SETOTHERMODE H, act on the low WORD in" mode besides " of the display processor 500.

[0109] G TEXTURE: [Table 5]

This command turns on / turns off texture mapping, gives a texture coordinate scaling, and chooses the number of tiles (inside of the texture stretched). A scale parameter is the format of (.16) and carries out enlarging or contracting of the texture parameter in a peak command. ON/OFF of a texture turn on / turn off texture coordinate processing of a geometrical pipeline. The number of tiles is equivalent to the tile chosen in a pipeline's raster section. The number of tiles holds the maximum level about details level (LOD) (MIDDO mapping).

[0110] G LOOKAT X: [Table 6]

This command is used for automatic generating of a texture coordinate. This command describes the directivity of an eye so that a signal processor 400 can know about what should generate a texture coordinate. The XYZ value (decimal with a 8-bit sign which has a 7-bit decimal) has described the vector in world space (space between a model view matrix and a projection matrix). This space is perpendicular to a televiewer's ***** direction, and it has turned [space] to a televiewer's right-hand side.

[0111] G Although it is the same as LOOKAT Y:G LOOKAT X, the zero WORD of the beginning in the segment by which addressing was carried out is zero (0x00000000).

[0112] If [generating of DP command] drawing 20 is referred to again, when the following display list command will turn to the display processor 500, a signal processor 400 only writes a command in a display processor (block 626 of drawing 20).

Block 626 carries out direct memory access of the display processor command to the display processor 500 through the X bus 218, or sets a display processor command to the buffer in main memory 300 for access by the display processor.

[0113] [Matrix command] When the following display list command is a matrix command, a signal processor 400 updates the state of the present matrix currently used (drawing 20, block 628), and arranges the updated matrix to a matrix stack (block 630). As mentioned above, in this example, a signal processor 400 maintains ten deep modeling / viewing matrix stacks. A new matrix can be loaded on a stack, and multiplication is carried out in the upper part of a stack (connected), or pop-off is carried out from a stack. In this example, the signal processor 400 holds "1 ** deep" projection matrix. therefore -- although a new matrix can be loaded to the present matrix or multiplication can be carried out -- a push -- or it cannot carry out pop

[0114] In this example, modeling / viewing matrix stack exists in main memory 300. The video game program 108 must assign sufficient memory for this stack, and must give a pointer to the stack area in a task list 250. The format of a matrix is optimized by the vector unit 420 of a signal processor. In order to offer suitable resolution, in this example, a signal processor 400 expresses each matrix value with 32 bits "double precision." 16 bits of high orders are assigned to an integral part with a sign (a part for larger value part than 1 is shown), and 16 bits of low ranks are assigned to the decimal part (a part for the value part between 0 and 1 is shown). However, the vector unit 420 of this example operates with a 16-bit wide value, and cannot carry out the multiplication of the 32-bit wide value directly. A matrix format (drawing 21 (b) what was shown) carries out grouping of all the integral parts of an element, and it carries out grouping of all the decimal parts of an element continuously. Thereby, a signal processor 400 can process a matrix more efficiently by carrying out the multiplication of a 16 bits integral part and the 16-bit decimal part separately, without repeating and carrying out an "unpack" or the "pack" of the matrix.

[0115] For example, the vector unit 420 can carry out the multiplication of each of the integral value with a 16-bit fixed-point sign of a matrix line by a certain operation, and can carry out the multiplication of each of the 16-bit decimal part of the same line by another operation. The result of these two portions is made into one, and a 32-bit double precision value is acquired, or the result (as opposed to the operation only whose integral part of a result requires only the decimal part of a result) of two portions is used separately. In this example, even when the vector unit 420 operates with a 16-bit value and it does not have clear "double precision" capacity, a matrix display carries out possible [of a signal processor 400 processing a 32 bit precision

value efficiently]. The followings are a signal processor matrix command and the example of a format of relation.

[0116] [The example of a matrix command]

G MTX: [Table 7]

This matrix command is the method controlled by the flag of a parameter field, and specifies the 4x4 transformation matrix (refer to drawing 21 (b)) used in order to change the continuing geometry. Length is a matrix byte's size which carries out ingress. The 4x4 transformation matrix specified by this command has the following formats. It is the block with which memory continued and is ROW. The element of 16 of a matrix is included in order of MAJOR. Each element of a matrix is the form of the fixed-point format S15.16. The length of 4x4 matrices in a byte is 64 bytes. Since the address of the actual matrix of main memory 300 is constituted, Segment id and an address field are used (refer to G SEGMENTSP, when information is more nearly required).

[0117] The following flags are used in a parameter field.

G Discriminate a MTX MODELVIEW ingress matrix as a model view matrix needed for performing efficient conversion of the polygonal method line for shading etc. (at the time of a default)

G Discriminate a MTX PROJECTION ingress matrix as a projection matrix which does not influence conversion of the polygonal method line for shading etc.

G A MTX MUL ingress matrix is connected in the present upper part of a matrix stack. (at the time of a default)

G A MTX LOAD ingress matrix replaces the present upper part of a matrix (model view or projection) stack.

G The present upper part of a MTX NOPUSH matrix stack is not pushed before performing loading or connection operation in the upper part of a stack. (at the time of a default)

G The present upper part of a MTX PUSH matrix stack is pushed before performing loading or connection operation in the upper part of a stack. A push is only supported by G MTX MODELVIEW, and is not supported by G MTX PROJECTION. This is for there to be no projection matrix stack (for you to have to reload projection clearly).

[0118] This single command that has the combination of a parameter enables the matrix operation used for usual [various kinds of]. For example, (G MTX LOAD|G MTX NOPUSH) replaces the upper part of a stack. (G MTX MUL|G MTX PUSH) performs connection, and makes a stack a general modeling layered structure.

[0119] A polygonal normal must be changed by reverse transposition of a model view

matrix to a lighting and a texture ring (refer to "OpenGL programming guide"). This is because a separate model view and a projection stack must be maintained and an ingress matrix must be discriminated.

[0120] G POPMTX: [Table 8]

This command carries out pop [of the stack of a model view matrix]. A parameter field is 0. It does not carry out pop [of the empty stack]. Since there is no stack of a projection matrix, this command is supported only to a model view matrix.

[0121] G VIEWPORT: [Table 9]

This command sends view port structure to a graphics pipeline. Since the address of the main memory 300 of actual view port structure is constituted, Segment id and an address field are used (refer to G SEGMENT, when information is more nearly required).

[0122] View port conversion is scale conversion of a normalization screen coordinate. Generally, since the requirements for the hardware of a screen device coordinate are suited, a view port must be constituted in collaboration with a projection matrix. The scale to x and y and the term of conversion have the 2-bit decimal required to suit positioning of the sub pixel in hardware. z value does not have a decimal.

[0123] By counting a decimal bit using one of the projection matrices of a default, view port structure can carry out initial setting in this way.

[Equation 1]

[0124] Again, with reference to [processing of peak command] drawing 20, if the following display list command is a "peak command", a signal processor 400 will change the peak as for which was pinpointed by this peak command in the present matrix state, and the shade was carried out as much as possible by the present lighting state, and will perform a clipping test about the peak, and the obtained peak is loaded to 408 in data memory 404. The signal processor 400 is equipped with the peak buffer holding the peak to 16 pieces in this example. Drawing 22 (a) shows the peak buffer of a signal processor 400 fully exposed by the main processor 100 and the video game program 108. This internal peak buffer 408 that can hold even 16 points is memorized by the data memory 404 of a signal processor, and is read by the main processor 100.

[0125] Although the signal processor 400 of this example can treat only a line, a triangle, or a rectangle (Sir face which is got blocked and defined by 2, 3, or 4 vertices), the vertex buffer 408 of this example memorizes the vertex to 16 pieces. Therefore, a signal processor 400 can carry out the reuse of the changed peak value instead of re-calculating the peak to whenever [the]. In this example, 3D permission / modeling

software used for a ***** sake in the video game program 108 organize a display list 110 the optimal, in order to make the reuse (and speed performance) of the peak into the maximum.

[0126] Since each peak memorized by the peak buffer 408 is expressed, drawing 22 (b) shows the example of the peak data structure which a signal processor 400 uses. In this example, the value of x corresponding to the peak changed, and y, z and w is memorized in double precision form, and a decimal part exists following an integral part (field 408(1) (a) -408(1), (h)). A peak color (r, g, b, alpha) is memorized by field 408(1) (i) -408(1) and (l), and a peak texture coordinate (s, t) is memorized by the field 408 (1), (m), and 408 (1) and (n). Furthermore, the peak value (what was got blocked, was changed and was projected on the viewing flat surface) of the coordinate of screen space is memorized by field 408(1) (o) -408(1) and (t) from this example (1 / w value is memorized in double precision form). A screen coordinate is used by the display processor 500 in order to draw the polygon defined by the vertex. The changed 3-dimensional coordinate is held at the peak buffer 408 for a clipping test. Since a polygon (it is not a vertex) is clipped and the reuse of the vertex in the vertex buffer 408 is carried out to a multiplex polygon, these changed 3D vertex values are memorized for clipping which should be performed and in which multiplication is possible. Furthermore, the peak data structure 408 (1) has the flag 408 (1) which can be used since a signal processor 400 specifies a clip test result (is it got blocked, and is the peak inside [each] six different clip flat surfaces, or is it outside?), and (v). The fluoroscopy-projection coefficient memorized by 408 (1) and (t) is held the field 408 (1) and (s) for fluoroscopy-coordinate operation performed by the texture coordinate unit (it explains below) of a display processor.

[0127] The following is one example of the peak command format used since it loads to the internal peak buffer which has some points.

G VTX: [Table 10]

This command loads the point (N+1) to the vector buffer which begins from the location v0 in a peak buffer. Since the actual VTX structure of the address of main memory 300 is constituted, Segment id and an address field are used (when information is still more nearly required, G SEGMENT-refer to). In order to make it possible to express the all 16 peaks by 4 bits, several n of the peak is encoded as "minus 1." Length is 16 times the number of the points, and is the size of VTX structure (from byte). A peak coordinate is 16 binary integers and the texture coordinates s and t are S10.5. A flag is disregarded in this example. The peak has the color or the normal (shading sake). A color is a 8-bit unsigned integer. A normal is a

decimal with a 8-bit sign (7-bit decimal). (The map of the 0x7f is carried out to +1.0, and the map of 0x81 is carried out to -1.0, and the map of 0x0 is carried out to 0.0). It must normalize, i.e., a normal vector is $\text{root}(x^2+y^2+z^2) \leq 127$. [0128] In case a peak command is received, a signal processor 400 changes the peak which used the present modeling / viewing matrix, and was pinpointed by the peak command (drawing 20, block 632). Refer to Chapter 3 ("viewing") of an OpenGL programming guide (silicon graphics 1993) besides Neider. These conversion carries out orientation of the object expressed by the peak in 3-dimensional space in proportion to the selected view. For example, in proportion to the selected view, these conversion carries out the parallel displacement of the object currently displayed, rotates and/or carries out enlarging or contracting. Such conversion calculation forces it the use which a burden requires for the vector unit 420 of a signal processor, and its capacity, in order to perform eight parallel computing simultaneously. In this example, the changed result is memorized by peak data structure field 408(1) (a) -408(1) and (h) in double precision form.

[0129] The [clip test] signal processor 400 determines whether next a clip test (drawing 20, block 636) is performed, and there is any changed peak inside a scene, or it is outside. Six clipping flat surfaces define the side and end of viewing volume. The changed each peak is compared with each of these six flat surfaces, and the result (that is, in which side of a clip flat surface is the peak located?) of comparison is memorized in the "flag" field 408 of a peak buffer (v) (refer to drawing 22 (b)). These results answer a "triangular command" and are used by the clipping block 646 (see the following). Since this example clips a polygon and does not clip a vertex, it is an observing point for the block 636 of drawing 20 not to perform clipping in fact, but to only test [as opposed to / a clip flat surface / only] a vertex position.

[0130] Next a projection matrix is used for [projection] signal processor 400, and it changes a peak value (drawing 20, block 638). The purpose of projection conversion is defining the viewing volume used by two methods. Viewing volume is determining how an object is projected on a two-dimensional viewing screen (it is got blocked and perspective or the right ** method is used). a shortening property -- having -- or (when the projection matrix defines the right ** method) -- right -- the peak of the result changed into target practice is projected on the two-dimensional viewing flat surface from 3-dimensional space (when the projection matrix defines perspective) (see the 90 page of an OpenGL programming guide or subsequent ones.) These coordinate values are written in a peak buffer data structure in field 408(1) (o) -408(1) and (t) (1/"w" value is held for next projection correction).

[0131] The [lighting] signal processor 400 performs a lighting operation, in order "to illuminate" each peak pinpointed by the peak command next. The video game system 50 is supporting many complicated real-time light effects including environmental (uniform) light, diffusion (directivity) light, and (texture mapping was used) a mirror-plane highlight. According to the lighting instruction execution in this example, a signal processor 400 loads overlay of the SP microcode 108 first, in order to perform a lighting operation. G The SETGEOMETRYMODE command must be specified and the above-mentioned G NUM LIGHTS command must define for light that the lighting operation was made usable. Usually, although the portion of the microcode 108 which performs a lighting operation does not exist in a signal processor 400, when a lighting call is performed, it is inputted into a signal processor 400 through overlay. A certain object is illuminated and this has close relation for the performance to a rendering scene in the state where other objects were colored statically. An over-write [this example / lighting overlay / a clipping microcode] in order to realize best performance which was most suitable for making the object which the illuminated scene clipped into the minimum, or losing it completely.

[0132] In order to carry out the lighting of the object, the peak which constitutes an object must be replaced with the specified color, and must be a normal. A normal is constituted from this example by three 8 numbers of bits showing x of the normal, y, and z component with a sign (refer to the above-mentioned G VTX command format). Each component is in the range of the value of -128 to +127 in this example. x components are equivalent to the position of the red of the peak, and, in y component, z component corresponds to blue green. Alpha still has no change. A normal vector must be normalized as mentioned above.

[0133] When an object changes the sense, by changing how an object appears, a lighting also realizes the effect of depth. In this example, a signal processor 400 supports the diffused light to seven in a scene. Each light has the direction and the color. Illuminating each light is continued in the same (as opposed to opening "a world") direction until it is not concerned with the sense of an object and a ** person but the direction of light is changed. Furthermore, one ambient light takes out uniform lighting. Shading is not simply supported in this example at a signal processor 400.

[0134] As mentioned above, lighting information is sent to a signal processor 400 by the optical data structure. the number of the diffused lights -- 0 to 7 -- **** -- things are made The variable of the value of red, green, and blue expresses the color of light, and takes the value of the range of 0 to 255. Subscript x and the variable which has y and z express the direction of light. It is decided by the direction that light

is shown. It means that the direction where a direction [as opposed to / this / light in the direction of light] is shown, and light is illuminating it is not shown (for example, a direction is set to $x=-141$, $y=-141$, and $z=0$ when light enters from the upper left direction of a world). In order to eliminate ambient light, a programmer has to specify that ambient light is black (0, 0, 0).

[0135] G The light command is used in order to make the set of the light on a display list active. Once light is activated, such light remains until the light of the following set is activated. The setup of the optical structure where this is new means an over-write [the old optical structure in a signal processor 400]. In order to turn ON calculation of light so that light may demonstrate an effect, it is required to turn ON a lighting mode bit using the G SETGEOMETRYMODE command. The above-mentioned lighting structure is used in order to offer the value of the color memorized to peak buffer field 408(1) (i) -408(1) and (l).

[0136] Next, a texture coordinate scaling / [generation] signal processor 400 performs a texture coordinate scaling and/or generation (drawing 20, block 642). In this example, the operation performed by block 642 realizes mirror-plane highlight, reflective mapping, and environmental mapping. in order to acquire these effects, in this example, a co-processor 200 uses light or the texture map of an environmental picture, and calculates the texture coordinate s and t based on the angle to a Sir face normal from a view The need of calculating a Sir face normal by each pixel by this texture mapping method although a mirror-plane lighting is realized can be abolished. It is too much [in calculation] severe for the video game system 50 in this example to calculate a Sir face normal by each pixel.

[0137] The mirror-plane highlight from almost all light is expressed by the texture map which defines a round-head dot with the exponential function or gauss function which shows intensity distribution. When the scene includes the highlight by the light of others, such as a fluorescent lamp or a sword which shines, fabricated specially, there is no difficulty in the rendering which can realize the texture map of a highlight.

[0138] In this example, although the display processor 500 performs texture mapping operation, a signal processor 400 performs texture coordinate transformation about each peak, when these effects are required. It is specified with the value in the G SETGEOMETRYMODE command a signal processor's 400 un-operating [an operation or] (see the above-mentioned publication). [of texture coordinate transformation] Furthermore, the G SETGEOMETRYMODE command specifies the linearization of the generated texture coordinate, in order to enable use of a panorama texture map, when performing for example, environmental mapping.

[0139] In this example, in generating of the texture coordinate of a signal processor 400, in order to refer to a texture and to derive s and t index, respectively, projection of the peak normal of the x and the direction of y of screen space is performed. Since s and t are generated, the angle between the views and Sir face normals in each peak is used. In this example, in order to realize actual s and t value, enlarging or contracting of the normal projection is carried out. A signal processor 400 maps the peak "behind" a view in 0, and maps positive projection in a scale value.

[0140] In this example, a texture ring operates using the G TEXTURE command which the attribute command of a signal processor 400 mentioned above by the way. This command gives the scale value which performs the above-mentioned texture coordinate mapping out of other things.

[0141] As mentioned above, texture coordinate mapping performed by the signal processor 400 needs the information which specifies the sense of a view in this example so that the angle between the Sir face normal of the peak and a view can be calculated. G The LOOKAT X and G LOOKAT Y commands give the sense of the view in automatic generating of the texture coordinate performed by the signal processor 400. The changed texture coordinate value is memorized by the field 408 of a peak data structure (1), (m), and 408 (1) and (n) by the signal processor 400, when calculated. These texture coordinate values are supplied to the display processor 500, in order to perform obtained texture mapping using the texture specified by the G TEXTURE command.

[0142] Since texture mapping is used for these effects, they cannot be used for the object by which texture mapping is carried out by the option.

[0143] [Writing which is a peak buffer] After performing all the steps of these, a signal processor 400 writes the peak value illuminated [was changed and] and projected in the peak buffer 408 (drawing 20, block 644), and returns to analyzing the following display list command (block 622).

[0144] Once a signal processor 400 writes [[triangular command-processing]] the peak in the peak buffer 408, a display list 110 can give a "triangular command." This "triangular command" specifies the polygon defined by the vertex in the vertex buffer 408. This "triangular command" is the demand to the signal processor 400 for generating the graphical display command showing a polygon, and sending this command to the display processor 500 fundamentally, for a rendering. In this example, a signal processor 400 gives the primitive of three different kinds, i.e., a line, a triangle, and a rectangle. In this example, in order to give a line or a triangle, it is required to load the module with which microcodes 108 differ. In this example, rectangles are

[no] the two-dimensional primitives specified by the screen coordinate, and clipping and a scissoring are carried out.

[0145] The following is one example of the format of a triangular command, and a related function.

The command below [the example of a triangular command] specifies the triangle defined by three vertices in a vertex buffer.

G TRI1: [Table 11]

This command produces one triangle using the vertices v0, v1, and v2 memorized by the internal vertex buffer. N field discriminates which [of the three peaks] contains the normal (flat shading sake) of a field (flat shading sake), or the color of a field.

[0146] The following commands control a signal processor 400, and since the command of the display processor 500 which carries out the rendering of the line defined by the two peaks in the peak buffer 408 is generated, they are used.

[0147] G LINE3D: [Table 12]

This command produces one line using the peaks v0 and v1 memorized by the internal peak buffer. N field specifies which [of the two peaks] contains the color of a field (flat shading sake).

[0148] A texture is carried out, and the filled rectangle requires the intervention of a signal processor 400, and, so, serves as operation of a signal processor. The following is the example of the command format of a texture rectangle command, and the function of relation.

[0149] G TEXTRECT: [Table 13]

[Table 14]

[Table 15]

These three commands draw 2D rectangle which has the present texture. Parameters x0 and y0 pinpoint the corner of rectangular upper left direction, and parameters x1 and y1 pinpoint the corner of lower right direction. All coordinates are 12 bits. S and T are the 10.5 numbers of bits with a sign, and specify the coordinate of the upper left direction of s and t. DsDx and DtDy are the 5.10 numbers of bits with a sign, and specify change of s (t) to change of x (y) coordinate.

[0150] In this example, a signal processor 400 supports the G TEXTRECT FLIP command which is the same as the G TEXTRECT command except for a texture being moved so that s coordinate may change in the direction of y or t coordinate may change in the x directions.

[0151] G FILLRECT: [Table 16]

This command draws 2D rectangle in the present fill color. Parameters x0 and y0

pinpoint the corner of rectangular upper left direction, and parameters x1 and y1 pinpoint the corner of lower right direction. All coordinates are 12 bits.

[0152] If it returns to clipping / [setup] drawing 20, a signal processor 400 will perform required clipping of the peak, if a triangular command is received (drawing 20, block 646). This clipping operation removes the portion of the geometrical primitive which exists in the outside of six clip flat surfaces which decide a viewing flat surface.

[0153] As mentioned above, about each peak, the result of the performed clip test 636 is memorized by the peak buffer 408, and can be used. When this triangular command shows the primitive decided by these peaks, a signal processor 400 advances this primitive clipping. When all the primitive peaks exist in the space decided by six clip flat surfaces, all primitives exist in an operating space and clipping is unnecessary. (It was shown by the flag field of the peak data structure 408 (1) shown in drawing 22 (b) like) When all the peaks that decide a primitive exist in the outside of the same clip flat surface, all primitives are eliminated from a display and thrown away. When a thing with the peak which decides a primitive exists in an operating space and a certain thing exists in the outside of an operating space on the other hand, a primitive needs to carry out clipping and the new peak is defined (or although all the peaks exist in the outside of an operating space, when deciding the primitive which passes this operating space). These tests and operations are performed by the clipping block 646 in this example.

[0154] Next, a signal processor 400 performs rear-face curling (drawing 20, block 647). It is determined that this operation will be on the rear face of an object, and a writing speed is made into the maximum by throwing away the polygon hidden from the visual field. this example — front facing — be -- back facing -- be -- the curling of both of the primitives is carried out by block 647 (got blocked and thrown away) The primitive type which should be carried out curling is specified with the parameter in the above-mentioned G SETGEOMETRYMODE command. A different curling flag which is put in order by this in the directions where geometry is arbitrary, or attains various effects is arranged in the places (for example, internal surface, a two-side polygon, etc.) used.

[0155] A signal processor 400 performs setup operation (drawing 20, block 648), and sends a graphical display command to the display processor 500, and it controls it so that the display processor 500 carries out the rendering of the primitive (drawing 20, block 650). In (block 648) and this example, a signal processor 400 is changed into the physical address for which the display processor 500 uses the address by which "segmentation" in a display list 110 was carried out as a part of setup operation (the

display processor 500 is a physical address machine in this example).

[0156] In this example, a signal processor 400 uses a segment table 416 (refer to drawing 23) so that it may support, in case addressing of the main memory 300 is carried out. The address in a signal processor 400 is shown more to a detail by table input 417a and 26-bit offset 417b. Refer for table input 417a to one of the base addresses of 16 in the segment address table 416. The base address referred to is added to offset 417b, and generates the physical address to main memory 300. A signal processor 400 constitutes the address of main memory 300 by adding a base address to a segment and (for example, given by the display list 110) 26-bit offset. A segment table 416 is constituted based on the G SEGMENT command of the following examples.

[0157] G SEGMENT: [Table 17]

This command applies an input to the above-mentioned segment table 416.

Segmentation addressing used by the signal processor 400 is effective in easy-izing animation by which the double buffer was carried out in this example. For example, the video game program 108 is in the state which gave the same offset as two different segments, and can hold two copies of the fragmentation of a certain display list in main memory 300. It is easy for the same grade as exchanging the segment pointer in a signal processor 400 to switch those copies. As another usage, grouping of data and the texture may be carried out into a certain segment, and grouping of the static background geometry may be carried out into another segment. Carrying out grouping of the data helps to optimize the memory cash advance in a main processor 100. All data including the embedding address must be preceded by the suitable G SEGMENT command which loads the right base address to the segment table of a signal processor 400.

[0158] Although the signal processor 400 is using the segment-addressing method shown in drawing 23, it cannot use this composition for the display processor 500 in this example. Therefore, a part of setup processing 648 must change the segment address which specifies the data structure needed for a rendering into the physical address which can carry out direct use by the display processor 500.

[0159] The main outputs of the signal processor 400 for [DP command write-in] graphics are one or more commands to drawing 20 and the display processor 500 outputted by block 650. Clipping is illuminated [although a main processor 100 (or video game storage 54) can supply the command of the display processor 500 directly, a signal processor 400 is changed to 3D picture, and it is projected, and] and carried out, and since the display processor command expressing the primitive by which

curling was carried out is generated, generally to perform the above-mentioned transform processing is needed.

[0160] The repertory of a display processor command is shown in drawing 102 from drawing 65. A signal processor 400 corresponds to including information and address information suitable in carrying out former TINGU of the display processor command generated appropriately, and a command. Furthermore, a signal processor 400 can generate and output the suitable mode and the attribute command which are needed for carrying out the rendering of the specific primitive as which the display processor was specified by the signal processor 400 using the suitable parameter (the mode to the display processor 500 and many of attribute commands are usually directly supplied by the main processor 100 under control of the video game program 108). As mentioned above, although a main processor 100 can offer the display processor 500 directly, since a certain display processor command is generated at least always when 3D object should be transformed, generally it needs to depend on a signal processor.

[0161] [Flow control command processing] It returns to drawing 20 again, and when the display list command received from the signal processor 400 is a flow control command, a signal processor 400 answers this command by the suitable method, and use that is, refer to the display list 110 for it in detail. A command and the following examples of a format give a flow control.

[0162] [The example of a flow control command]

G DL: [Table 18]

This command is used in order to specify another display list and to generate the hierarchy of a display list, a nest display list, indirection reference, etc. The segment field discriminates a memory segment. An address field is the offset from the base of a segment. Moreover, these form the address in the main memory 300 which specifies a new display list. In this example, although it is desirable to end all display lists by the G ENDDL command, the length field (not shown) can describe the length of a new display list from a byte. The parameter field holds the flag which controls the moving state of a transfer. When Flag G DL NOPUSH is set, the present display list is not pushed on a stack before transfer control. This achieves **** like BRANCH or a GOTO instruction rather than a hierarchization display list (this is effective in decomposing a larger display list into discontinuous memory piece, next connecting them by branching of a display list).

[0163] G ENDDL: [Table 19]

And a display list command ends branching of the hierarchy of a display list, and produces "pop" in processing of the hierarchy of a display list. Since [which is

referred to as being ended by the command instead of giving the length of a display list deductively / changing] it is got blocked and the display list piece of strange size is constituted, this command is the most effective. All display lists are ended by this command.

[0164] G NOOP: [Table 20]

None of this command is set up. This command is internally generated under a certain situation.

[0165] Drawing 20 and block 652 maintain the stack of the display list in main memory 300, and perform the function which pushes and carries out the wildebeest ping (crossing) of the stack of this display list. Block 652 will stop a signal processor 400, if a signal processor 400 meets with an "open end" display list command.

[0166] [microcode audio processing of a signal processor] -- in addition to the graphics operation mentioned above, the signal processor 400 of this example performs digital audio processing. The vector unit 420 of a signal processor 400 is suitable for especially performing "sum of product" calculation. Especially this "sum of product" calculation is effective in digital signal processing of audio signals, such as for example, audio compression release, wave table re-sampling, composition, and filtering, of a certain kind. Since the accumulator of 48-bit width of face is contained in the vector unit data path 423, the digital space and/or frequency filtering using many taps can be considerably adapted, without spoiling precision. As one example of the optimal specific usage of the vector unit 420 for audio processing, eight separate register files 422 and related data paths 423 of the vector unit 420 of a signal processor 400 can be used in order to process eight different MTDI audios in parallel simultaneously. The following is the example of the further audio processing which can be effectively performed using the vector unit 420.

- The wave table composition using - solid interpolation which can process in parallel, the -8 ** audio, i.e., the 8 times as many sample as this, which can solve a polynomial, and here Four of the data paths 423 of the vector unit 420 are used in order to process one sample. the data path 423 of other four vector units 420 Here [- audio envelope processing and here] where it is used in order to process the 2nd sample. The data path 423 of eight vector units 420. The data path 423 of eight vector units 420 can carry out the multiplication of the mixer weighting factor corresponding to an audio sample different, respectively - audio mixing processing which can carry out the multiplication of the weighting factor which is different to an audio sample different, respectively, and here. [0167] a signal processor 400 requires what [1/] of the audio reproduction fruit time interval concerned, in order to perform digital audio processing

relevant to the audio reproduction real-time interval and to complete, since audio digital signal processing can be performed effectively at high speed. For example, in order for a signal processor 400 to process an audio for digital one, only time much fewer than $1 / 30$ seconds is needed, but the audio interface 208 of a co-processor reproduces this by the real time by $1/30$ of the second time intervals. In this example, time sharing of the signal processor 400 is carried out on account of this capacity between graphics operation and digital audio processing.

[0168] Generally, a main processor 100 gives a task list 250 to a signal processor 400 at the beginning of a video frame. This video frame specifies the picture and sound which should be generated in the video frame following a degree. You have to complete both the audio for the video frame to which a co-processor 200 will follow the degree concerned by the time the video frame following the degree concerned starts, and graphics operation. Since video presentation and audio reproduction are consecutive processing of the real time (it must be got blocked, and new video presentation must be supplied at each video frame time, and an audio must be supplied continuously), until, as for a co-processor 200, the following frame starts — each — it is necessary to end all the audios and video signal processings relevant to the video frame following a degree.

[0169] In this example, the signal processor 400 is shared by digital audio signal processing and graphics operation. On account of the high-speed calculation capacity of the vector unit 420 of a signal processor, it is time much shorter than the present video frame time, and a signal processor 400 can complete processing of the audio which should be reproduced between the video frames which continue at a degree, and it is time shorter than the present video frame time, and can also complete the graphics operation of the picture which should be displayed between the pictures following a degree. Thereby, a task list 250 can specify both the graphics display list which must be completed by a signal processor 400 and the co-processor 200 by the time the next video frame time starts, and an audio reproduction list. However, before the next video frame time starts [a main processor 100 / a co-processor 200] in a co-processor 200 in this example, what bars giving the task list 250 which cannot be completed does not have anything. It will not perform, by the time a signal processor 400 can continue processing of a task list about the whole present video frame time by the compound graphics and audio processing which are demanded by the signal processor 400 fully concentrating if time is this thing, and the following video frame starts. The video game program 108 should process too much burden by the suitable method, when it is made not to overwork a co-processor 200 and is worked hard. A

video game programmer can avoid too much burden of a signal processor 400 by considering so that the processing (for example, clipping) for which all the display lists 110 are organized efficiently, carry out the modeling of the object by the effective method by 3D, and include broadly, and time requires them may be avoided or it may be made the minimum. However, also by such consideration, a co-processor 200 needs the time more than one video frame time, in order to complete processing of a complicated picture especially. A video game programmer can solve this situation by making effective frame speed late so that a television set 58 may carry out regeneration of the same picture memorized in a part of frame buffer 118 between the multiplex video frames within a time on which a co-processor 200 can complete processing of the following picture. Since a user may perceive an adjustable frame speed as delay which is not desirable, in order to complete the picture which processing concentrates most, it is often the best to reduce the whole effective frame speed even at the speed of which it is required more for a co-processor 200 (a thereby more complicated picture can protect appearing slowly from the picture which is not more complicated).

[0170] Since a user will hear obstructive "click" sound about audio processing in the stream of the audio which others followed, failing to supply an audio between predetermined video frame time is being unable to admit generally. Disturbance of such an audio is [that it is easy to go into an ear] noisy. Therefore, they should avoid. One method of avoiding the audio disturbance whose signal processor 400 rings the assigned audio processing in the ear under the situation that it has not completed to within a time is ordering to reproduce the audio in which a main processor's 100 is equivalent to a front frame between the frames which follow the audio interface 208 at a degree. If it carries out carefully, a desirable audio can be generated in this way, without a user noticing disturbance. Other methods are making a signal processor 400 process the audio equivalent to two or more video frames within one video frame time. Thereby, a different (early) effective audio "frame" speed from effective video frame speed can be supplied. "Effective frame speed" means the speed which a co-processor 200 makes generate the information equivalent to one frame (in this example, an actual video frame speed of a television set is fixed).

[Translation done.]

*** NOTICES ***

Japan Patent Office is not responsible for any damages caused by the use of this translation.

1.This document has been translated by computer. So the translation may not reflect the original precisely.

2.**** shows the word which can not be translated.

3.In the drawings, any words are not translated.

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]

[Drawing 1] The whole video game system which generates 3D picture and digital processing stereo sound is shown.

[Drawing 2] The example of the 3D screen effect realizable using the system of drawing 1 is shown.

[Drawing 3] The example of the 3D screen effect realizable using the system of drawing 1 is shown.

[Drawing 4] One example of the main elements of a whole video game system is shown.

[Drawing 5] One example of main processing operation of a whole video game system is shown.

[Drawing 6] One example of whole operation of a video game system is shown.

[Drawing 7] In order to generate a graphic picture, one example of all the steps performed by the video game system is shown.

[Drawing 8] One example of detailed whole system architecture is shown.

[Drawing 9] One example of the initialization routine of a main processor is shown.

[Drawing 10] One example of the memory map of a main processor is shown.

[Drawing 11] One example of the internal architecture of a co-processor is shown.

[Drawing 12] One example of the architecture of the internal bus of a co-processor is shown.

[Drawing 13] One example of the internal architecture of a signal processor is shown.

[Drawing 14] (a) shows one example of an instruction format of a signal processor, and (b) shows the source concerned of (a) for processing by the vector unit shown in

drawing 13, and certain 1 of operation of the addition which is and shows one example of slicing of the lameness point field and by which (c) is performed by the example of the vector unit of a signal processor example.

[Drawing 15] One example of the register of a signal processor is shown.

[Drawing 16] One example of the register of a signal processor is shown.

[Drawing 17] One example of a hierarchy task list including a graphical display list and an audio reproduction list is shown.

[Drawing 18] One example of a microcode loading routine is shown.

[Drawing 19] One example of processing of the display list of the illustrated easy signal processor is shown.

[Drawing 20] One example of the control step sequence of the graphic microcode of a signal processor is shown.

[Drawing 21] (a) shows one example of double precision expression, and (b) shows one example of a matrix format.

[Drawing 22] (a) shows one example of a peak buffer format of a signal processor, and (b) shows one example of a definition of peak data.

[Drawing 23] One example of the composition of segment addressing of a signal processor is shown.

[Drawing 24] One example of the architecture of audio software is shown.

[Drawing 25] One example of processing of the reproduction list of easy signal processors is shown.

[Drawing 26] One example of the control step sequence of the audio microcode of a signal processor is shown.

[Drawing 27] One example of the audio processing structure of a signal processor is shown.

[Drawing 28] One example of the whole display processor processing step is shown.

[Drawing 29] One example of the pipeline structure of a display processor is shown.

[Drawing 30] One example of the architecture of a display processor is shown.

[Drawing 31] One example of the register of a display processor is shown.

[Drawing 32] One example of the register of a display processor is shown.

[Drawing 33] One example of the register of a display processor is shown.

[Drawing 34] One example of the register of a display processor is shown.

[Drawing 35] One example of the composition of the tile descriptor of texture memory is shown.

[Drawing 36] One example of texture unit processing is shown.

[Drawing 37] One example of the architecture of a texture coordinate unit and a

texture memory unit is shown.

[Drawing 38] One example of the lookup in the Color Index mode of texture memory is shown.

[Drawing 39] One example of more detailed use of the texture memory for memorizing a Color Index texture is shown.

[Drawing 40] One example of operation of a color combiner is shown.

[Drawing 41] One example of operation of an alpha combiner is shown.

[Drawing 42] One example of operation of an alpha fix rise is shown.

[Drawing 43] One example of a primitive blend of a different type is shown.

[Drawing 44] One example of operation of a blender is shown.

[Drawing 45] One example of a format of a color pixel is shown.

[Drawing 46] One example of a format of a depth pixel is shown.

[Drawing 47] One example of the generation processing which can be written in is shown.

[Drawing 48] One example of the architecture of a video interface is shown.

[Drawing 49] One example of the operating sequence of a video interface is shown.

[Drawing 50] One example of the control register of a video interface is shown.

[Drawing 51] One example of the control register of a video interface is shown.

[Drawing 52] One example of the control register of a video interface is shown.

[Drawing 53] One example of the architecture of a main memory interface is shown.

[Drawing 54] One example of the control register of a memory interface is shown.

[Drawing 55] One example of the control register of a memory interface is shown.

[Drawing 56] One example of the architecture of the interface of a main processor is shown.

[Drawing 57] One example of the register of the interface of a memory processor is shown.

[Drawing 58] One example of the architecture of an audio interface is shown.

[Drawing 59] One example of the register of an audio interface is shown.

[Drawing 60] One example of the architecture of a serial interface is shown.

[Drawing 61] One example of the register of a serial interface is shown.

[Drawing 62] One example of the architecture of a peripheral interface adapter is shown.

[Drawing 63] One example of control/status register of a peripheral interface adapter is shown.

[Drawing 64] One example of control/status register of a peripheral interface adapter is shown.

[Drawing 65] The format and the related function of "the set of a color picture" of a graphics viewing command in a display processor are shown.

[Drawing 66] The format and the related function of "the set of a texture picture" of a graphics viewing command in a display processor are shown.

[Drawing 67] The format and the related function of "the set of Z picture" of a graphics viewing command in a display processor are shown.

[Drawing 68] The format and the related function of "the set of a tile" of a graphics viewing command in a display processor are shown.

[Drawing 69] The format and the related function of "loading of a tile" of a graphics viewing command in a display processor are shown.

[Drawing 70] The format and the related function of "loading of a block" of a graphics viewing command in a display processor are shown.

[Drawing 71] The format and the related function of "the set of tile size" of a graphics viewing command in a display processor are shown.

[Drawing 72] The format and the related function of "loading of a tart" of a graphics viewing command in a display processor are shown.

[Drawing 73] The various types of a triangular command are shown.

[Drawing 74] The format and the related function of "the coefficient of an edge" of a graphics viewing command in a display processor are shown.

[Drawing 75] The format and the related function of "the coefficient of an edge" of a graphics viewing command in a display processor are shown.

[Drawing 76] The format and the related function of "the coefficient of an edge" of a graphics viewing command in a display processor are shown.

[Drawing 77] The format and the related function of "the coefficient of the shade" of a graphics viewing command in a display processor are shown.

[Drawing 78] The format and the related function of "the coefficient of the shade" of a graphics viewing command in a display processor are shown.

[Drawing 79] The format and the related function of "the coefficient of a texture" of a graphics viewing command in a display processor are shown.

[Drawing 80] The format and the related function of "the coefficient of a texture" of a graphics viewing command in a display processor are shown.

[Drawing 81] The format and the related function of "the coefficient of Z-uffer" of a graphics viewing command in a display processor are shown.

[Drawing 82] The format and the related function of "the rectangular fill" of a graphics viewing command in a display processor are shown.

[Drawing 83] The format and the related function of "the rectangular texture" of a

graphics viewing command in a display processor are shown.

[Drawing 84] The format and the related function of "the texture of a rectangle flip" of a graphics viewing command in a display processor are shown.

[Drawing 85] The format and the related function of "the set in combined-harvester-and-thresher mode" of a graphics viewing command in a display processor are shown.

[Drawing 86] The format and the related function of a set" in the mode of a graphics viewing command in a display processor besides " are shown.

[Drawing 87] The format and the related function of a set" in the mode of a graphics viewing command in a display processor besides " are shown.

[Drawing 88] The format and the related function of "the set of an environmental color" of a graphics viewing command in a display processor are shown.

[Drawing 89] The format and the related function of "the set of a primitive color" of a graphics viewing command in a display processor are shown.

[Drawing 90] The format and the related function of "the set of a blend color" of a graphics viewing command in a display processor are shown.

[Drawing 91] The format and the related function of "the set of a fog color" of a graphics viewing command in a display processor are shown.

[Drawing 92] The format and the related function of "the set of a fill color" of a graphics viewing command in a display processor are shown.

[Drawing 93] The format and the related function of "the set of primitive depth" of a graphics viewing command in a display processor are shown.

[Drawing 94] The format and the related function of "the set of a scissor" of a graphics viewing command in a display processor are shown.

[Drawing 95] The format and the related function of "the set of conversion" of a graphics viewing command in a display processor are shown.

[Drawing 96] The format and the related function of "the set of Key R" of a graphics viewing command in a display processor are shown.

[Drawing 97] The format and the related function of "the set of Key GB" of a graphics viewing command in a display processor are shown.

[Drawing 98] The format and the related function of the "full sink" of a graphics viewing command in a display processor are shown.

[Drawing 99] The format and the related function of the "load sink" of a graphics viewing command in a display processor are shown.

[Drawing 100] The format and the related function of the "pipe sink" of a graphics viewing command in a display processor are shown.

[Drawing 101] The format and the related function of the "tile sink" of a graphics viewing command in a display processor are shown.

[Drawing 102] The format and the related function of "having no operation" of a graphics viewing command in a display processor are shown.

[Description of Notations]

52 -- Main unit

100 -- Main processor

200 -- Co-processor

300 -- Main memory

400 -- Signal processor

500 -- Display processor

[Translation done.]

*** NOTICES ***

Japan Patent Office is not responsible for any damages caused by the use of this translation.

1.This document has been translated by computer. So the translation may not reflect the original precisely.

2.**** shows the word which can not be translated.

3.In the drawings, any words are not translated.

DRAWINGS

[Drawing 39]

[Drawing 45]

[Drawing 1]

[Drawing 2]

[Drawing 3]

[Drawing 9]

[Drawing 19]

[Drawing 4]

[Drawing 5]

[Drawing 6]
[Drawing 18]
[Drawing 25]
[Drawing 7]
[Drawing 10]
[Drawing 16]
[Drawing 8]
[Drawing 11]
[Drawing 12]
[Drawing 17]
[Drawing 21]
[Drawing 13]
[Drawing 14]
[Drawing 20]
[Drawing 33]
[Drawing 15]
[Drawing 22]
[Drawing 23]
[Drawing 24]
[Drawing 26]
[Drawing 27]
[Drawing 28]
[Drawing 35]
[Drawing 46]
[Drawing 29]
[Drawing 30]
[Drawing 31]
[Drawing 32]
[Drawing 34]
[Drawing 37]
[Drawing 55]
[Drawing 36]
[Drawing 38]
[Drawing 40]
[Drawing 51]
[Drawing 41]

[Drawing 42]
[Drawing 43]
[Drawing 64]
[Drawing 44]
[Drawing 47]
[Drawing 48]
[Drawing 49]
[Drawing 50]
[Drawing 52]
[Drawing 54]
[Drawing 53]
[Drawing 56]
[Drawing 57]
[Drawing 67]
[Drawing 58]
[Drawing 59]
[Drawing 60]
[Drawing 61]
[Drawing 62]
[Drawing 63]
[Drawing 65]
[Drawing 66]
[Drawing 68]
[Drawing 69]
[Drawing 71]
[Drawing 73]
[Drawing 70]
[Drawing 72]
[Drawing 74]
[Drawing 75]
[Drawing 76]
[Drawing 77]
[Drawing 81]
[Drawing 78]
[Drawing 79]
[Drawing 92]

[Drawing 80]
[Drawing 82]
[Drawing 84]
[Drawing 83]
[Drawing 85]
[Drawing 86]
[Drawing 90]
[Drawing 87]
[Drawing 88]
[Drawing 89]
[Drawing 91]
[Drawing 93]
[Drawing 98]
[Drawing 94]
[Drawing 95]
[Drawing 96]
[Drawing 97]
[Drawing 99]
[Drawing 100]
[Drawing 101]
[Drawing 102]

[Translation done.]

【外國語明細書】

1. Title of Invention

High Performance Low Cost Video Game System With Coprocessor Providing High Speed Efficient 3D Graphics And Digital Audio Signal Processing

2. Claims

1. An interactive video game system comprising:

an interactive user input device;

a main processor coupled to the input device, the main processor having an address space, the main processor interactively selecting a point of view in response to inputs from the user input device;

a coprocessor coupled to the main processor, the coprocessor providing a predetermined graphics feature set for interactively generating image data in response to the selected point of view by projecting polygons representing a three dimensional world onto a two dimensional viewing plane, the coprocessor including:

a signal processor that is shared between at least graphics functions and audio processing functions, the signal processor including a scalar unit and a vector unit, the vector unit capable of performing plural calculations in parallel, the signal processor including a microcode store that stores microcode, the signal processor executing the microcode in the microcode store to perform the graphics and audio processing functions;

a display processor comprising display pipeline hardware that alternatively provides a one-pixel-per-cycle mode and a two-pixel-per-cycle mode to minimize hardware while providing a rich feature set including level-of-detail processing, the display pipeline hardware including a texture memory having first and second parts, the texture memory first part being capable of storing texture maps that are color indexed and texture maps that are not color indexed, the texture memory second part being capable of storing texture maps and/or color lookup tables for the color indexed texture maps,

a video interface,

整理番号=000568

(2)

an audio interface,
a serial interface, and
a parallel peripheral interface,

wherein each of the signal processor, the display processor, the video interface, the audio interface, the serial interface and the parallel peripheral interface includes circuitry for accessing a main memory;

the main memory being coupled to the coprocessor via a 9 bit wide bus, the main memory providing a common address space for the coprocessor and the main processor, the main memory storing at least the following data structures:

Instructions for execution by the main processor;
a color frame buffer;
a depth buffer;
graphics microcode;
audio processing microcode;
at least one display list;
at least one texture map; and
at least one audio output buffer;

a video signal generating circuit coupled to the coprocessor video interface, the video signal generating circuit generating a video signal for display on a color television set;

a removable storage device including a housing, a security chip, a read only memory and at least one further memory device, the coprocessor including an arrangement that maps the read only memory and the further memory device into the main processor address space, the read only memory initially storing the graphics and audio processing microcode; and

a connector that connects the coprocessor to the removable storage device; and
a serial peripheral interface circuit coupled to the coprocessor serial interface, the serial peripheral interface circuit including a processor that performs serial interface functions and security functions and further includes a boot ROM that provides main

整理番号= 0 0 0 5 6 8

(3)

processor initial program load instructions, the serial interface circuit processor being coupled to the removable storage device security chip through the connector.

2. An interactive real time graphics display system comprising:

at least one user input device;

a main random access memory providing a common address space;

a main processor coupled to address the main memory and also coupled to the user input device, the main processor storing instructions in and executing instructions from the main memory in real time response to inputs received from the user input device, the main processor storing at least display list graphics commands and play list audio commands in the main memory;

a signal processor coupled to address the main memory, the signal processor fetching and executing microcode stored in the main memory, the signal processor reading the display list graphics commands and play list audio commands from the main memory, the signal processor generating audio sample data in response to the play list audio commands and generating graphics display commands in response to the display list, the signal processor storing the sample data in an audio output buffer allocated within the main memory;

a display processor coupled to address the main memory, the display processor generating image data based at least in part on at least one texture map and other graphics data stored in the main memory, the display processor producing the image data in response to the graphics display commands, the display processor storing the image data in a color image frame buffer within the main memory;

a video interface coupled to address the main memory, the video interface reading the color image frame buffer in synchronism with display raster scan; and

an audio interface coupled to address the main memory, the audio interface reading the audio output buffer in synchronism with real time sound generation.

整理番号=000568

(4)

3. A method of operating a graphics display system of the type including a main processor, a coprocessor coupled to the main processor, a main random access memory coupled to the coprocessor and addressable by both the main processor and the coprocessor, and a video signal generating arrangement that produces a video signal for display, the method including the following steps:
- (a) storing main processor code into the main memory;
 - (b) executing, with the main processor, the main processor code stored by the storing step, said executing step including storing coprocessor code, a task list, at least one texture map and a color lookup table into the main memory;
 - (c) fetching the task list from main memory;
 - (d) processing the task list with the coprocessor in accordance at least in part with the coprocessor code stored by step (b), the processing step including performing the following steps:
 - (1) loading the texture map and the color lookup table from the main memory into an on-chip texture memory;
 - (2) performing at least one 3D geometric transformation on a set of vertices using a scalar unit and a vector unit including performing multiple calculations in parallel with the vector unit;
 - (3) generating a triangle command based on the 3D geometric transformation;
 - (4) generating a pixel value in response to the triangle command;
 - (5) accessing the texture memory twice to provide color indexed texels based on the triangle command;
 - (6) combining the texels with the generated pixel value to generate a combined pixel value;
 - (7) accessing pixel values in a frame buffer stored in the main memory;
 - (8) blending the combined pixel value with at least one pixel value stored in the frame buffer;

整理番号=000568

(5)

- (9) conditionally writing the combined pixel value into the frame buffer based on a comparison using a depth buffer stored in the main memory;
 - (10) using said scalar and vector units to generate output audio samples including performing multiple calculations in parallel with the vector unit; and
 - (11) storing the output audio samples into the main memory;
 - (e) reading the frame buffer in real time synchronism with color television set line scanning and converting the frame buffer contents to a composite video signal; and
 - (f) reading the stored output audio samples in real time and converting the stored audio samples into stereo sound.
4. A process for generating at least one display mode control command for processing by a 3D graphics system, the process including the step of generating at least one set mode command having:
- a command identifier field including a six-bit binary value of 101111, and
 - at least one of the following mode control fields:
 - (k) an atomic primitive mode field that specifies whether to force writing a primitive to a frame buffer before reading a following primitive,
 - (i) a cycle type mode field that selects a display pipeline cycle control mode,
 - (h) a perspective texture enable mode field that selectively enables perspective texture correction,
 - (g) a texture detail mode field that selectively enables texture detail processing,
 - (f) a texture sharpen enable mode field that selectively enables texture sharpening,
 - (e) a texture detail enable mode field that selectively enables texture level-of-detail processing,
 - (d) an enable look up table mode field that selectively enables lookup of texture values from a color look up table,
 - (c) a texture look up table type mode field that specifies type of texels in the color look up table,

整理番号=000568

(6)

- (b) a sample type mod field that specifies how texels should be sampled,
- (a) a mid texel mode field that specifies whether texels should be filtered using a 2×2 half texel interpolation,
- (Z) a first bilerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 0,
- (Y) a second bilerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 1,
- (X) a texel convert mode field that specifies whether a texel outputted by the texture filter during pipeline cycle 0 should be color converted,
- (W) a chroma key enable mode field that selectively enables chroma keying,
- (V2) an rgb dither select mode field that selects type of rgb dithering,
- (V1) an alpha dither select mode field that selects type of alpha dithering,
- (V) a plurality of blend modewords that specify blender parameters,
- (M) a force blend enable mode field that specifies whether the blender should be force enabled,
- (L) an alpha coverage select mode field that specifies whether coverage should be used to determine pixel alpha,
- (K) a coverage times alpha select mode field that specifies whether coverage multiplied by alpha should be used to determine pixel alpha and coverage,
- (J) a z mode select mode field that specifies z buffering mode,
- (I) a coverage destination mode field that specifies coverage destination,
- (H) a color on coverage mode field that specifies whether color should be updated only on coverage overflow,
- (G) an image read enable mode field that selectively enables color and/or coverage read/modify/write frame buffer memory access,
- (F) a z update enable mode field that selectively enables z buffer writing conditioned on whether color write is enabled,
- (E) a z compare enable mode field that specifies conditional color write enable on depth comparison,

整理番号=0 0 0 5 6 8

(7)

- (D) an anti-alias enable mode field that allows blend enable using coverage,
 - (C) a z source select mode field that chooses between primitive depth and pixel depth.
 - (B) a dither alpha enable mode field that specifies whether random noise should be used in alpha compare, and
 - (A) an alpha compare enable mode field that enables conditional color write on alpha compare.
5. A process as in claim 1 including the step of generating a cycle type mode field that selects a display pipeline cycle control mode of 1 cycle per pixel mode, 2 cycles per pixel mode, copy mode and fill mode.
6. A process as in claim 1 including the step of generating a texture look up table type mode field that between:
- (1) storing texels in a color look up table in RGBA format of 5 bits red, 5 bits green, 5 bits blue and 1 bit alpha, and
 - (2) storing texels in a color look up table in intensity alpha format providing an 8 bit intensity value and an 8 bit alpha value.
7. A process as in claim 1 including the step of generating a sample type mode field that selects between:
- (1) point sampling, and
 - (2) 2×2 array sampling.
8. A process as in claim 1 including the step of generating an rgb dither select mode field that selects between dithering based on:
- (1) a magic square matrix,
 - (2) a bayer matrix,
 - (3) noise, or

整理番号=000568

(8)

- (4) no dithering.
9. A process as in claim 1 including the step of generating an alpha dithering select mode field that specifies dithering based on:
- (1) a predetermined pattern,
 - (2) the negative of the predetermined pattern,
 - (3) noise, or
 - (4) no dithering.
10. A process as in claim 1 including the step of generating a plurality of blend modeword that specify blender parameters specifying:
- selectively multiplying a first blender input during pipeline cycle 0,
 - selectively multiplying the first blender input during pipeline cycle 1,
 - selectively multiplying a second blender input during pipeline cycle 0,
 - selectively multiplying the second blender input during pipeline cycle 1,
 - selectively multiplying a third blender input during pipeline cycle 0,
 - selectively multiplying the third blender input during pipeline cycle 1,
 - selectively multiplying a fourth blender input during pipeline cycle 0,
 - selectively multiplying the fourth blender input during pipeline cycle 1.
11. A process as in claim 1 including the step of generating a coverage destination mode field that selects between the following coverage destination modes:
- (1) clamp,
 - (2) wrap,
 - (3) force to full coverage, and
 - (4) save.
12. A process as in claim 1 including the step of generating a z mode select mode field that selects one of the following z buffering modes:

整理番号=000568

(9)

- (1) opaque,
 - (2) interpenetrating,
 - (3) transparent, and
 - (4) decal.
13. A system for generating at least one 3D display mode control command for processing by a 3D graphics system, the system including:
- at least one processor,
 - at least one memory, and
 - circuitry coupled to the processor and to the memory for providing at least one set mode command having:
 - a command identifier field including a six-bit binary value of 101111, and
 - at least one of the following mode control fields:
 - (k) an atomic primitive mode field that specifies whether to force writing a primitive to a frame buffer before reading a following primitive,
 - (i) a cycle type mode field that selects a display pipeline cycle control mode,
 - (h) a perspective texture enable mode field that selectively enables perspective texture correction,
 - (g) a texture detail mode field that selectively enables texture detail processing,
 - (f) a texture sharpen enable mode field that selectively enables texture sharpening,
 - (e) a texture detail enable mode field that selectively enables texture level-of-detail processing,
 - (d) an enable look up table mode field that selectively enables lookup of texture values from a color look up table,
 - (c) a texture look up table type mode field that specifies type of texels in the color look up table,
 - (b) a sample type mode field that specifies how texels should be sampled,

整理番号=0 0 0 5 6 8

(10)

- (a) a mid texel mode field that specifies whether texels should be filtered using a 2×2 half texel interpolation,
- (Z) a first bilerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 0,
- (Y) a second bilerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 1,
- (X) a texel convert mode field that specifies whether a texel outputted by the texture filter during pipeline cycle 0 should be color converted,
- (W) a chroma key enable mode field that selectively enables chroma keying,
- (V2) an rgb dither select mode field that selects type of rgb dithering,
- (V1) an alpha dither select mode field that selects type of alpha dithering,
- (V) a plurality of blend modewords that specify blender parameters,
- (M) a force blend enable mode field that specifies whether the blender should be force enabled,
- (L) an alpha coverage select mode field that specifies whether coverage should be used to determine pixel alpha,
- (K) a coverage times alpha select mode field that specifies whether coverage multiplied by alpha should be used to determine pixel alpha and coverage,
- (J) a z mode select mode field that specifies z buffering mode,
- (I) a coverage destination mode field that specifies coverage destination,
- (H) a color on coverage mode field that specifies whether color should be updated only on coverage overflow,
- (G) an image read enable mode field that selectively enables color and/or coverage read/modify/write frame buffer memory access,
- (F) a z update enable mode field that selectively enables z buffer writing conditioned on whether color write is enabled,
- (E) a z compare enable mode field that specifies conditional color write enable on depth comparison,
- (D) an anti-alias enable mode field that allows blend enable using coverage,

整理番号=000568

(11)

(C) a z source select mode field that chooses between primitive depth and pixel depth.

(B) a dither alpha enable mode field that specifies whether random noise should be used in alpha compare, and

(A) an alpha compare enable mode field that enables conditional color write on alpha compare.

14. A system as in claim 10 including means for generating a cycle type mode field that selects a display pipeline cycle control mode of 1 cycle per pixel mode, 2 cycles per pixel mode, copy mode and fill mode.
15. A system as in claim 10 including means for providing a texture look up table type mode field that selects between:
 - (1) storing texels in a color look up table in RGBA format of 5 bits red, 5 bits green, 5 bits blue and 1 bit alpha, and
 - (2) storing texels in a color look up table in intensity alpha format providing an 8 bit intensity value and an 8 bit alpha value.
16. A system as in claim 10 including circuitry for providing a sample type mode field that selects between:
 - (1) point sampling, and
 - (2) 2×2 array sampling.
17. A system as in claim 10 including circuitry for providing an rgb dither select mode field that selects between dithering based on:
 - (1) a magic square matrix,
 - (2) a bayer matrix,
 - (3) noise, or
 - (4) no dithering.

整理番号=000568

(12)

18. A system as in claim 10 including means for generating an alpha dithering select mode field that specifies dithering based on:
- (1) a predetermined pattern,
 - (2) the negative of the predetermined pattern,
 - (3) noise, or
 - (4) no dithering.
19. A system as in claim 10 including means for generating a plurality of blend modeword that specify blender parameters specifying:
- selectively multiplying a first blender input during pipeline cycle 0,
 - selectively multiplying the first blender input during pipeline cycle 1,
 - selectively multiplying a second blender input during pipeline cycle 0,
 - selectively multiplying the second blender input during pipeline cycle 1,
 - selectively multiplying a thlrd blender input during pipellne cycle 0,
 - selectively multiplying the third blender input during pipeline cycle 1,
 - selectively multiplying a fourth blender input during pipeline cycle 0,
 - selectively multiplying the fourth blender input during pipeline cycle 1.
20. A system as in claim 10 including circuitry for generating a coverage destination mode field that selects between the following coverage destination modes:
- (1) clamp,
 - (2) wrap,
 - (3) force to full coverage, and
 - (4) save.
21. A system as in claim 10 including circultry for providing a z mode select mode field that selects one of the following z buffering modes:
- (1) opaque,
 - (2) interpenetrating,

整理番号=000568

(13)

- (3) transparent, and
 - (4) decal.
22. In a 3D graphics system, a process for interpreting at least one set mode command including the steps of:
- (1) interpreting a command identifier field including a six-bit binary value of 101111,
 - (2) interpreting at least one of the following mode control fields:
 - (k) an atomic primitive mode field that specifies whether to force writing a primitive to a frame buffer before reading a following primitive,
 - (i) a cycle type mode field that selects a display pipeline cycle control mode,
 - (h) a perspective texture enable mode field that selectively enables perspective texture correction,
 - (g) a texture detail mode field that selectively enables texture detail processing,
 - (f) a texture sharpen enable mode field that selectively enables texture sharpening,
 - (e) a texture detail enable mode field that selectively enables texture level-of-detail processing,
 - (d) an enable look up table mode field that selectively enables lookup of texture values from a color look up table,
 - (c) a texture look up table type mode field that specifies type of texels in the color look up table,
 - (b) a sample type mode field that specifies how texels should be sampled,
 - (a) a mid texel mode field that specifies whether texels should be filtered using a 2×2 half texel interpolation,
 - (Z) a first bilerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 0,
 - (Y) a second bilerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 1,

整理番号=000568

(14)

- (X) a texel convert mode field that specifies whether a texel outputted by the texture filter during pipeline cycle 0 should be color converted,
- (W) a chroma key enable mode field that selectively enables chroma keying,
- (V2) an rgb dither select mode field that selects type of rgb dithering,
- (V1) an alpha dither select mode field that selects type of alpha dithering,
- (V) a plurality of blend modewords that specify blender parameters,
- (M) a force blend enable mode field that specifies whether the blender should be force enabled,
- (L) an alpha coverage select mode field that specifies whether coverage should be used to determine pixel alpha,
- (K) a coverage times alpha select mode field that specifies whether coverage multiplied by alpha should be used to determine pixel alpha and coverage,
- (J) a z mode select mode field that specifies z buffering mode,
- (I) a coverage destination mode field that specifies coverage destination,
- (H) a color on coverage mode field that specifies whether color should be updated only on coverage overflow,
- (G) an Image read enable mode field that selectively enables color and/or coverage read/modify/write frame buffer memory access,
- (F) a z update enable mode field that selectively enables z buffer writing conditioned on whether color write is enabled,
- (E) a z compare enable mode field that specifies conditional color write enable on depth comparison,
- (D) an anti-alias enable mode field that allows blend enable using coverage,
- (C) a z source select mode field that chooses between primitive depth and pixel depth,
- (B) a dither alpha enable mode field that specifies whether random noise should be used in alpha compare,
- (A) an alpha compare enable mode field that enables conditional color write on alpha compare, and

整理番号=000568

(15)

- (3) generating an image based at least in part on step (2).
23. A process as in claim 19 including the step of interpreting a cycle type mode field that selects a display pipeline cycle control mode of 1 cycle per pixel mode, 2 cycles per pixel mode, copy mode and fill mode.
24. A process as in claim 19 including the step of interpreting a texture look up table type mode field that selects between:
- (1) storing texels in a color look up table in RGBA format of 5 bits red, 5 bits green, 5 bits blue and 1 bit alpha, and
 - (2) storing texels in a color look up table in intensity alpha format providing an 8 bit intensity value and an 8 bit alpha value.
25. A process as in claim 19 including the step of interpreting a sample type mode field that selects between:
- (1) point sampling, and
 - (2) 2×2 array sampling.
26. A process as in claim 19 including the step of interpreting an rgb dither select mode field that selects between dithering based on:
- (1) a magic square matrix,
 - (2) a bayer matrix,
 - (3) noise, or
 - (4) no dithering.
27. A process as in claim 19 including the step of interpreting an alpha dithering select mode field that specifies dithering based on:
- (1) a predetermined pattern,
 - (2) the negative of the predetermined pattern,

整理番号= 0 0 0 5 6 8

(16)

- (3) noise, or
 - (4) no dithering.
28. A process as in claim 19 including the step of interpreting a plurality of blend modeword that specify blender parameters specifying:
- selectively multiplying a first blender input during pipeline cycle 0,
 - selectively multiplying the first blender input during pipeline cycle 1,
 - selectively multiplying a second blender input during pipeline cycle 0,
 - selectively multiplying the second blender input during pipeline cycle 1,
 - selectively multiplying a third blender input during pipeline cycle 0,
 - selectively multiplying the third blender input during pipeline cycle 1,
 - selectively multiplying a fourth blender input during pipeline cycle 0,
 - selectively multiplying the fourth blender input during pipeline cycle 1.
29. A process as in claim 19 including the step of interpreting a coverage destination mode field that selects between the following coverage destination modes:
- (1) clamp,
 - (2) wrap,
 - (3) force to full coverage, and
 - (4) save.
30. A process as in claim 19 including the step of interpreting a z mode select mode field that selects one of the following z buffering modes:
- (1) opaque,
 - (2) interpenetrating,
 - (3) transparent, and
 - (4) decal.

整理番号=000568

(17)

31. A 3D graphics system for interpreting at least one set mode command having a command identifier field including a six-bit binary value of 101111, the system including:
- a first decoder that interprets a command identifier field including a six-bit binary value of 101111,
 - (k) circuitry for interpreting an atomic primitive mode field that specifies whether to force writing a primitive to a frame buffer before reading a following primitive,
 - (i) circuitry for interpreting a cycle type mode field that selects a display pipeline cycle control mode,
 - (h) circuitry for interpreting a perspective texture enable mode field that selectively enables perspective texture correction,
 - (g) circuitry for interpreting a texture detail mode field that selectively enables texture detail processing,
 - (f) circuitry for interpreting a texture sharpen enable mode field that selectively enables texture sharpening,
 - (e) circuitry for interpreting a texture detail enable mode field that selectively enables texture level-of-detail processing,
 - (d) circuitry for interpreting an enable look up table mode field that selectively enables lookup of texture values from a color look up table,
 - (c) circuitry for interpreting a texture look up table type mode field that specifies type of texels in the color look up table,
 - (b) circuitry for interpreting a sample type mode field that specifies how texels should be sampled,
 - (a) circuitry for interpreting a mid texel mode field that specifies whether texels should be filtered using a 2×2 half texel interpolation,
 - (Z) circuitry for interpreting a first bilerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 0,
 - (Y) circuitry for interpreting a second bilerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 1,

整理番号=0 0 0 5 6 8

(18)

(X) circuitry for interpreting a texel convert mode field that specifies whether a texel outputted by the texture filter during pipeline cycle 0 should be color converted,

(W) circuitry for interpreting a chroma key enable mode field that selectively enables chroma keying,

(V2) circuitry for interpreting an rgb dither select mode field that selects type of rgb dithering,

(V1) circuitry for interpreting an alpha dither select mode field that selects type of alpha dithering,

(V) circuitry for interpreting a plurality of blend modewords that specify blender parameters,

(M) circuitry for interpreting a force blend enable mode field that specifies whether the blender should be force enabled,

(L) circuitry for interpreting an alpha coverage select mode field that specifies whether coverage should be used to determine pixel alpha,

(K) circuitry for interpreting a coverage times alpha select mode field that specifies whether coverage multiplied by alpha should be used to determine pixel alpha and coverage,

(J) circuitry for interpreting a z mode select mode field that specifies z buffering mode,

(I) circuitry for interpreting a coverage destination mode field that specifies coverage destination,

(H) circuitry for interpreting a color on coverage mode field that specifies whether color should be updated only on coverage overflow,

(G) circuitry for interpreting an image read enable mode field that selectively enables color and/or coverage read/modify/write frame buffer memory access,

(F) circuitry for interpreting a z update enable mode field that selectively enables z buffer writing conditioned on whether color write is enabled,

(E) circuitry for interpreting a z compare enable mode field that specifies conditional color write enable on depth comparison,

整理番号=000568

(19)

(D) circuitry for interpreting an anti-alias enable mode field that allows blend enable using coverage,

(C) circuitry for interpreting a z source select mode field that chooses between primitive depth and pixel depth,

(B) circuitry for interpreting a dither alpha enable mode field that specifies whether random noise should be used in alpha compare,

(A) circuitry for interpreting an alpha compare enable mode field that enables conditional color write on alpha compare, and

circuitry coupled to above-mentioned circuitry (a)-(k), (A)-(M), (V1), (V2) and (W)-(Z) for generating an image.

32. An apparatus as in claim 28 including means for interpreting a cycle type mode field that selects a display pipeline cycle control mode of 1 cycle per pixel mode, 2 cycles per pixel mode, copy mode and fill mode.

33. An apparatus as in claim 28 including means for interpreting a texture look up table type mode field that selects between:

(1) storing texels in a color look up table in RGBA format of 5 bits red, 5 bits green, 5 bits blue and 1 bit alpha, and

(2) storing texels in a color look up table in intensity alpha format providing an 8 bit intensity value and an 8 bit alpha value.

34. An apparatus as in claim 28 including means for interpreting a sample type mode field that selects between:

(1) point sampling, and

(2) 2×2 array sampling.

35. An apparatus as in claim 28 including means for interpreting an rgb dither select mode field that selects between dithering based on:

整理番号=000568

(20)

- (1) a magic square matrix,
 - (2) a bayer matrix,
 - (3) noise, or
 - (4) no dithering.
36. An apparatus as in claim 28 including means for interpreting an alpha dithering select mode field that specifies dithering based on:
- (1) a predetermined pattern,
 - (2) the negative of the predetermined pattern,
 - (3) noise, or
 - (4) no dithering.
37. An apparatus as in claim 28 including means for interpreting a plurality of blend modeword that specify blender parameters specifying:
- selectively multiplying a first blender input during pipeline cycle 0,
 - selectively multiplying the first blender input during pipeline cycle 1,
 - selectively multiplying a second blender input during pipeline cycle 0,
 - selectively multiplying the second blender input during pipeline cycle 1,
 - selectively multiplying a third blender input during pipeline cycle 0,
 - selectively multiplying the third blender input during pipeline cycle 1,
 - selectively multiplying a fourth blender input during pipeline cycle 0,
 - selectively multiplying the fourth blender input during pipeline cycle 1.
38. An apparatus as in claim 28 including means for interpreting a coverage destination mode field that selects between the following coverage destination modes:
- (1) clamp,
 - (2) wrap,
 - (3) force to full coverage, and

整理番号=000568

(21)

- (4) save.
39. An apparatus as in claim 28 including means for interpreting a z mode select mode field that selects one of the following z buffering modes:
- (1) opaque,
 - (2) interpenetrating,
 - (3) transparent, and
 - (4) decal.
40. A storage medium for use with a 3D graphics system, the storage medium storing at least one 3D display mode control command including:
- a command identifier field including a six-bit binary value of 101111, and
- at least one of the following mode control fields:
- (k) an atomic primitive mode field that specifies whether to force writing a primitive to a frame buffer before reading a following primitive,
 - (l) a cycle type mode field that selects a display pipeline cycle control mode,
 - (h) a perspective texture enable mode field that selectively enables perspective texture correction,
 - (g) a texture detail mode field that selectively enables texture detail processing,
 - (f) a texture sharpen enable mode field that selectively enables texture sharpening,
 - (e) a texture detail enable mode field that selectively enables texture level-of-detail processing,
 - (d) an enable look up table mode field that selectively enables lookup of texture values from a color look up table,
 - (c) a texture look up table type mode field that specifies type of texels in the color look up table,
 - (b) a sample type mode field that specifies how texels should be sampled,

整理番号=000568

(22)

(a) a mid texel mode field that specifies whether texels should be filtered using a 2×2 half texel interpolation,

(Z) a first blerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 0,

(Y) a second blerp mode field that specifies whether a texture filter should bilinearly interpolate texels in pipeline cycle 1,

(X) a texel convert mode field that specifies whether a texel outputted by the texture filter during pipeline cycle 0 should be color converted,

(W) a chroma key enable mode field that selectively enables chroma keying,

(V2) an rgb dither select mode field that selects type of rgb dithering,

(V1) an alpha dither select mode field that selects type of alpha dithering,

(V) a plurality of blend modewords that specify blender parameters,

(M) a force blend enable mode field that specifies whether the blender should be force enabled,

(L) an alpha coverage select mode field that specifies whether coverage should be used to determine pixel alpha,

(K) a coverage times alpha select mode field that specifies whether coverage multiplied by alpha should be used to determine pixel alpha and coverage,

(J) a z mode select mode field that specifies z buffering mode,

(I) a coverage destination mode field that specifies coverage destination,

(H) a color on coverage mode field that specifies whether color should be updated only on coverage overflow,

(G) an image read enable mode field that selectively enables color and/or coverage read/modify/write frame buffer memory access,

(F) a z update enable mode field that selectively enables z buffer writing conditioned on whether color write is enabled,

(E) a z compare enable mode field that specifies conditional color write enable on depth comparison,

(D) an anti-alias enable mode field that allows blend enable using coverage,

整理番号=0 0 0 5 6 8

(23)

(C) a z source select mode field that chooses between primitive depth and pixel depth.

(B) a dither alpha enable mode field that specifies whether random noise should be used in alpha compare, and

(A) an alpha compare enable mode field that enables conditional color write on alpha compare.

41. A storage medium as in claim 37 including means for storing a cycle type mode field that selects a display pipeline cycle control mode of 1 cycle per pixel mode, 2 cycles per pixel mode, copy mode and fill mode.

42. A storage medium as in claim 37 including means for storing a texture look up table type mode field that selects between:

(1) storing texels in a color look up table in RGBA format of 5 bits red, 5 bits green, 5 bits blue and 1 bit alpha, and

(2) storing texels in a color look up table in intensity alpha format providing an 8 bit intensity value and an 8 bit alpha value.

43. A storage medium as in claim 37 including means for storing a sample type mode field that selects between:

(1) point sampling, and

(2) 2×2 array sampling.

44. A storage medium as in claim 37 including means for storing an rgb dither select mode field that selects between dithering based on:

(1) a magic square matrix,

(2) a bayer matrix,

(3) noise, or

(4) no dithering.

整理番号=0 0 0 5 6 8

(24)

-
45. A storage medium as in claim 37 including means for storing an alpha dithering select mode field that specifies dithering based on:
- (1) a predetermined pattern,
 - (2) the negative of the predetermined pattern,
 - (3) noise, or
 - (4) no dithering.
46. A storage medium as in claim 37 including means for storing a plurality of blend modeword that specify blender parameters specifying:
- selectively multiplying a first blender input during pipeline cycle 0,
 - selectively multiplying the first blender input during pipeline cycle 1,
 - selectively multiplying a second blender input during pipeline cycle 0,
 - selectively multiplying the second blender input during pipeline cycle 1,
 - selectively multiplying a third blender input during pipeline cycle 0,
 - selectively multiplying the third blender input during pipeline cycle 1,
 - selectively multiplying a fourth blender input during pipeline cycle 0,
 - selectively multiplying the fourth blender input during pipeline cycle 1.
47. A storage medium as in claim 37 including means for storing a coverage destination mode field that selects between the following coverage destination modes:
- (1) clamp,
 - (2) wrap,
 - (3) force to full coverage, and
 - (4) save.
48. A storage medium as in claim 37 including means for storing a z mode select mode field that selects one of the following z buffering modes:
- (1) opaque,

整理番号=0 0 0 5 6 8

(25)

- (2) interpenetrating,
 - (3) transparent, and
 - (4) decal.
49. A process for generating at least one display mode control command for processing by a 3D graphics system, the process including the step of generating at least one set mode command having:
- a command Identifier field including a six-bit binary value of 111100, and
 - at least one of the following additional fields:
 - at least one combiner subtract mode control field that specifies subtracting at least one color space value from a color combiner,
 - at least one combiner multiply mode control field that specifies multiplying a color combiner input by at least one color space value, and
 - at least one combiner add control field that specifies a color combiner adder input.
50. A process as in claim 46 further including the step of generating the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a subtract Source A,
 - (2) a subtract source control field specifying a subtract Source B,
 - (3) a multiply source control field specifying a multiply source C, and
 - (4) an add source control field specifying an add source D.
51. A process as in claim 46 further including the step of generating the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a RGB component subtract Source A,
 - (2) a subtract source control field specifying an alpha component subtract Source A,

整理番号=0 0 0 5 6 8

(26)

- (3) a subtract source control field specifying a RGB component subtract Source B,
 - (4) a subtract source control field specifying an alpha component subtract Source B,
 - (5) a multiply source control field specifying an RGB component multiply source C,
 - (6) a multiply source control field specifying an alpha component multiply source C,
 - (7) an add source control field specifying an RGB component add source D, and
 - (8) an add source control field specifying an alpha component add source D.
52. A process as in claim 46 further including the step of generating the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 0;
 - (2) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 1;
 - (3) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 0;
 - (4) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 1;
 - (5) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 0;
 - (6) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 1;
 - (7) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 0;
 - (8) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 1;

整理番号=0 0 0 5 6 8(27)

(9) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 0;

(10) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 0;

(11) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 1;

(12) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 1;

(13) an add source control field specifying an RGB component add source D for pipeline cycle 0;

(14) an add source control field specifying an alpha component add source D for pipeline cycle 0;

(15) an add source control field specifying an RGB component add source D for pipeline cycle 1; and

(16) an add source control field specifying an alpha component add source D for pipeline cycle 1.

53. A process as in claim 46 further including:

generating at least first, second, third and fourth multiplexer select values specifying corresponding inputs for an RGB color combiner channel; and

generating at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for an alpha color combiner channel.

54. A process as in claim 46 further including:

generating at least first, second, third and fourth multiplexer select values specifying corresponding inputs for a pipeline cycle 0 color combine operation; and

generating at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for a pipeline cycle 1 color combine operation.

整理番号=000568

(28)

55. A system for generating at least one display mode control command for processing by a 3D graphics system, the system including:
- at least one processor,
 - at least one memory, and
 - circuitry coupled to the processor and to the memory for providing at least one set mode command having:
 - a command identifier field including a six-bit binary value of 111100, and
 - at least one of the following additional fields:
 - at least one combiner subtract mode control field that specifies subtracting at least one color space value from a color combiner,
 - at least one combiner multiply mode control field that specifies multiplying a color combiner input by at least one color space value, and
 - at least one combiner add control field that specifies a color combiner adder input.
56. A system as in claim 52 further including means for providing the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a subtract Source A,
 - (2) a subtract source control field specifying a subtract Source B,
 - (3) a multiply source control field specifying a multiply source C, and
 - (4) an add source control field specifying an add source D.
57. A system as in claim 52 further including means for generating the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a RGB component subtract Source A,
 - (2) a subtract source control field specifying an alpha component subtract Source A,
 - (3) a subtract source control field specifying a RGB component subtract Source B,

整理番号=0 0 0 5 6 8

(29)

- (4) a subtract source control field specifying an alpha component subtract Source B,
- (5) a multiply source control field specifying an RGB component multiply source C,
- (6) a multiply source control field specifying an alpha component multiply source C,
- (7) an add source control field specifying an RGB component add source D, and
- (8) an add source control field specifying an alpha component add source D.
58. A system as in claim 52 further including circuitry for generating the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 0;
- (2) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 1;
- (3) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 0;
- (4) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 1;
- (5) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 0;
- (6) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 1;
- (7) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 0;
- (8) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 1;
- (9) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 0;

整理番号=0 0 0 5 6 8

(30)

(10) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 0;

(11) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 1;

(12) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 1;

(13) an add source control field specifying an RGB component add source D for pipeline cycle 0;

(14) an add source control field specifying an alpha component add source D for pipeline cycle 0;

(15) an add source control field specifying an RGB component add source D for pipeline cycle 1; and

(16) an add source control field specifying an alpha component add source D for pipeline cycle 1.

59. A system as in claim 52 further including:

circuitry for providing at least first, second, third and fourth multiplexer select values specifying corresponding inputs for an RGB color combiner channel; and

means for providing at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for an alpha color combiner channel.

60. A system as in claim 52 further including:

circuitry for generating at least first, second, third and fourth multiplexer select values specifying corresponding inputs for a pipeline cycle 0 color combine operation; and

circuitry for generating at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for a pipeline cycle 1 color combine operation.

61. In a 3D graphics display system, a process for interpreting at least one set mode command comprising:

整理番号=000568

(31)

- (a) interpreting a command identifier field including a six-bit binary value of 111100,
- (b) interpreting at least one of the following additional fields:
- at least one combiner subtract mode control field that specifies subtracting at least one color space value from a color combiner,
 - at least one combiner multiply mode control field that specifies multiplying a color combiner input by at least one color space value, and
 - at least one combiner add control field that specifies a color combiner adder input, and
- (c) generating an image based at least in part on step (b).
62. A process as in claim 58 further including the step of interpreting the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a subtract Source A,
 - (2) a subtract source control field specifying a subtract Source B,
 - (3) a multiply source control field specifying a multiply source C, and
 - (4) an add source control field specifying an add source D.
63. A process as in claim 58 further including the step of interpreting the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a RGB component subtract Source A,
 - (2) a subtract source control field specifying an alpha component subtract Source A,
 - (3) a subtract source control field specifying a RGB component subtract Source B,
 - (4) a subtract source control field specifying an alpha component subtract Source B,
 - (5) a multiply source control field specifying an RGB component multiply source C,

整理番号=0 0 0 5 6 8

(32)

- (6) a multiply source control field specifying an alpha component multiply source C,
- (7) an add source control field specifying an RGB component add source D, and
- (8) an add source control field specifying an alpha component add source D.
64. A process as in claim 58 further including the step of interpreting the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 0;
- (2) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 1;
- (3) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 0;
- (4) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 1;
- (5) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 0;
- (6) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 1;
- (7) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 0;
- (8) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 1;
- (9) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 0;
- (10) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 0;

整理番号= 0 0 0 5 6 8

(33)

- (11) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 1;
- (12) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 1;
- (13) an add source control field specifying an RGB component add source D for pipeline cycle 0;
- (14) an add source control field specifying an alpha component add source D for pipeline cycle 0;
- (15) an add source control field specifying an RGB component add source D for pipeline cycle 1; and
- (16) an add source control field specifying an alpha component add source D for pipeline cycle 1.

65. A process as in claim 58 further including:

- interpreting at least first, second, third and fourth multiplexer select values specifying corresponding inputs for an RGB color combiner channel; and
- interpreting at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for an alpha color combiner channel.

66. A process as in claim 58 further including:

- interpreting at least first, second, third and fourth multiplexer select values specifying corresponding inputs for a pipeline cycle 0 color combine operation; and
- interpreting at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for a pipeline cycle 1 color combine operation.

67. A 3D graphics display system for interpreting at least one set mode command including a six-bit binary value of 111100, the system comprising:

- (a) means for interpreting a command Identifier field including a six-bit binary value of 111100,

整理番号=000568

(34)

(b) means for generating at least one control signal based on at least one of the following additional fields:

at least one combiner subtract mode control field that specifies subtracting at least one color space value from a color combiner,

at least one combiner multiply mode control field that specifies multiplying a color combiner input by at least one color space value, and

at least one combiner add control field that specifies a color combiner adder input, and

(c) means for generating an image based at least in part on the control signal.

68. An apparatus as in claim 64 further including a combiner circuit that interprets the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:

- (1) a subtract source control field specifying a subtract Source A,
- (2) a subtract source control field specifying a subtract Source B,
- (3) a multiply source control field specifying a multiply source C, and
- (4) an add source control field specifying an add source D.

69. An apparatus as in claim 64 further including a combiner that interprets the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:

- (1) a subtract source control field specifying a RGB component subtract Source A,
- (2) a subtract source control field specifying an alpha component subtract Source

A,

- (3) a subtract source control field specifying a RGB component subtract Source B,
- (4) a subtract source control field specifying an alpha component subtract Source

B,

- (5) a multiply source control field specifying an RGB component multiply source C,

整理番号=000568

(35)

- (6) a multiply source control field specifying an alpha component multiply source C,
- (7) an add source control field specifying an RGB component add source D, and
- (8) an add source control field specifying an alpha component add source D.
70. An apparatus as in claim 64 further including a combiner that combines color signals based on the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 0;
- (2) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 1;
- (3) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 0;
- (4) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 1;
- (5) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 0;
- (6) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 1;
- (7) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 0;
- (8) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 1;
- (9) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 0;
- (10) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 0;

整理番号=000568

(36)

- (11) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 1;
- (12) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 1;
- (13) an add source control field specifying an RGB component add source D for pipeline cycle 0;
- (14) an add source control field specifying an alpha component add source D for pipeline cycle 0;
- (15) an add source control field specifying an RGB component add source D for pipeline cycle 1; and
- (16) an add source control field specifying an alpha component add source D for pipeline cycle 1.

71. An apparatus as in claim 64 further including:

means for interpreting at least first, second, third and fourth multiplexer select values specifying corresponding inputs for an RGB color combiner channel; and

means for interpreting at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for an alpha color combiner channel.

72. An apparatus as in claim 64 further including:

means for interpreting at least first, second, third and fourth multiplexer select values specifying corresponding inputs for a pipeline cycle 0 color combine operation; and

means for interpreting at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for a pipeline cycle 1 color combine operation.

73. A storage medium for use with a 3D graphics system, the storage medium storing at least one display mode control command having:

a command identifier field including a six-bit binary value of 111100, and

at least one of the following additional fields:

整理番号=000568

(37)

at least one combiner subtract mode control field that specifies subtracting at least one color space value from a color combiner,

at least one combiner multiply mode control field that specifies multiplying a color combiner input by at least one color space value, and

at least one combiner add control field that specifies a color combiner adder input.

74. A storage medium as in claim 70 further including means for storing the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:

- (1) a subtract source control field specifying a subtract Source A,
- (2) a subtract source control field specifying a subtract Source B,
- (3) a multiply source control field specifying a multiply source C, and
- (4) an add source control field specifying an add source D.

75. A storage medium as in claim 70 further including means for storing the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:

- (1) a subtract source control field specifying a RGB component subtract Source A,
- (2) a subtract source control field specifying an alpha component subtract Source A,
- (3) a subtract source control field specifying a RGB component subtract Source B,
- (4) a subtract source control field specifying an alpha component subtract Source B,
- (5) a multiply source control field specifying an RGB component multiply source C,
- (6) a multiply source control field specifying an alpha component multiply source C,
- (7) an add source control field specifying an RGB component add source D, and
- (8) an add source control field specifying an alpha component add source D.

整理番号=000568

(38)

76. A storage medium as in claim 70 further including means for storing the following combiner subtract mode control fields to specify multiplexer sources for implementing the function $(A-B)*C + D$:
- (1) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 0;
 - (2) a subtract source control field specifying a RGB component subtract Source A for pipeline cycle 1;
 - (3) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 0;
 - (4) a subtract source control field specifying an alpha component subtract Source A for pipeline cycle 1;
 - (5) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 0;
 - (6) a subtract source control field specifying a RGB component subtract Source B for pipeline cycle 1;
 - (7) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 0;
 - (8) a subtract source control field specifying an alpha component subtract Source B for pipeline cycle 1;
 - (9) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 0;
 - (10) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 0;
 - (11) a multiply source control field specifying an RGB component multiply source C for pipeline cycle 1;
 - (12) a multiply source control field specifying an alpha component multiply source C for pipeline cycle 1;
 - (13) an add source control field specifying an RGB component add source D for pipeline cycle 0;

整理番号=000568

(39)

(14) an add source control field specifying an alpha component add source D for pipeline cycle 0;

(15) an add source control field specifying an RGB component add source D for pipeline cycle 1; and

(16) an add source control field specifying an alpha component add source D for pipeline cycle 1.

77. A storage medium as in claim 70 further including:

means for storing at least first, second, third and fourth multiplexer select values specifying corresponding inputs for an RGB color combiner channel; and

means for storing at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for an alpha color combiner channel.

78. A storage medium as in claim 70 further including:

means for storing at least first, second, third and fourth multiplexer select values specifying corresponding inputs for a pipeline cycle 0 color combine operation; and

means for storing at least fifth, sixth, seventh and eighth multiplexer select values specifying corresponding inputs for a pipeline cycle 1 color combine operation.

79. A process for generating at least one color image mode command for processing by a 3D graphics system, the process including the step of generating at least one command including:

a command identifier field including a six-bit binary value within the set of 111111 and 111101;

an image data format parameter;

a color element size parameter;

an image width parameter; and

a base address parameter.

整理番号=000568

(40)

80. A process as in claim 76 further including the step of generating the image data format parameter selecting between the following:
- (a) rgba,
 - (b) yuv,
 - (c) color index,
 - (d) intensity alpha, and
 - (e) alpha.
81. The process as in claim 76 further including the step of generating the color element size parameter selecting between the following:
- (a) 4 bit wide color element value,
 - (b) 8 bit wide color element value,
 - (c) 16 bit wide color element value, and
 - (d) 32 bit wide color element value.
82. The process as in claim 76 further including the step of generating the image width parameter value specifying the width in pixels of an image stored in memory.
83. The process as in claim 76 further including the step of generating the base address parameter specifying the base address in main memory of the top left corner of the image.
84. A system for providing at least one color image mode command for processing by a 3D graphics system, the system including
- at least one processor,
 - at least one memory, and
 - circuitry coupled to the processor and to the memory for generating at least one command including:

整理番号=000568

(41)

a command identifier field including a six-bit binary value within the set of 111111 and 111101;

an image data format parameter;

a color element size parameter;

an image width parameter; and

a base address parameter.

85. A system as in claim 81 further including means for providing the image data format parameter selecting between the following:
- (a) rgba,
 - (b) yuv,
 - (c) color index,
 - (d) intensity alpha, and
 - (e) alpha.
86. The system as in claim 81 further including means for providing the color element size parameter selecting between the following:
- (a) 4 bit wide color element value,
 - (b) 8 bit wide color element value,
 - (c) 16 bit wide color element value, and
 - (d) 32 bit wide color element value.
87. The system as in claim 81 further including means for providing the image width parameter value specifying the width in pixels of an image stored in memory.
88. The system as in claim 81 further including means for generating the base address parameter specifying the base address in main memory of the top left corner of the image.

整理番号=000568

(42)

89. In a 3D graphics system, a process of executing at least one color image mode command including:
- (a) interpreting a command identifier field including a six-bit binary value within the set of 111111 and 111101 as corresponding to color image mode;
 - (b) interpreting at least one of the following:
 - an image data format parameter;
 - a color element size parameter;
 - an image width parameter; and
 - a base address parameter; and
 - (c) generating a color image display based at least in part on step (b).
90. A process as in claim 86 further including the steps of selecting between the following based on the image data format parameter:
- (a) rgba,
 - (b) yuv,
 - (c) color index,
 - (d) intensity alpha, and
 - (e) alpha.
91. The process as in claim 86 further including the step of selecting between the following based on the color element size parameter:
- (a) 4 bit wide color element value,
 - (b) 8 bit wide color element value,
 - (c) 16 bit wide color element value, and
 - (d) 32 bit wide color element value.
92. The process as in claim 86 further including the step of setting the width in pixels of an image stored in memory based on the image width parameter value.

整理番号=000568

(43)

93. The process as in claim 86 further including the step of setting the base address in main memory of the top left corner of the image based on the base address parameter.
94. A 3D graphics system for executing at least one color image mode command having a command identifier field including a six-bit binary value within the set of 111111 and 111101 as corresponding to color image mode, the system including:
a command identifier decoder for interpreting a command identifier field including a six-bit binary value within the set of 111111 and 111101 as corresponding to color image mode;
a command parameter decoder for interpreting at least one of the following:
 an image data format parameter;
 a color element size parameter;
 an image width parameter; and
 a base address parameter; and
a display circuit that generates a color image display based at least in part on parameters.
95. An apparatus as in claim 91 further including structure for selecting between the following based on the image data format parameter:
(a) rgba,
(b) yuv,
(c) color index,
(d) intensity alpha, and
(e) alpha.
96. The process as in claim 91 further including structure for selecting between the following based on the color element size parameter:
(a) 4 bit wide color element value,

整理番号=000568

(44)

- (b) 8 bit wide color element value,
 - (c) 16 bit wide color element value, and
 - (d) 32 bit wide color element value.
97. The process as in claim 91 further including means for setting the width in pixels of an image stored in memory based on the image width parameter value.
98. The process as in claim 91 further including a circuit for setting the base address in main memory of the top left corner of the image based on the base address parameter .
99. A storage medium for use with a 3D graphics system, the storage medium for storing at least one color image mode command including:
a command identifier field including a six-bit binary value within the set of 111111 and 111101;
an image data format parameter;
a color element size parameter;
an image width parameter; and
a base address parameter.
100. A storage medium as in claim 96 further including means for storing an image data format parameter selecting between the following:
- (a) rgba,
 - (b) yuv,
 - (c) color index,
 - (d) intensity alpha, and
 - (e) alpha.

整理番号=0 0 0 5 6 8(45)

101. The storage medium as in claim 96 further including means for storing a color element size parameter selecting between the following:
- (a) 4 bit wide color element value,
 - (b) 8 bit wide color element value,
 - (c) 16 bit wide color element value, and
 - (d) 32 bit wide color element value.
102. The storage medium as in claim 96 further including means for storing an image width parameter value specifying the width in pixels of an image stored in memory.
103. The storage medium as in claim 96 further including means for storing the base address parameter specifying the base address in main memory of the top left corner of the image.
104. A process for generating at least one mask image mode command for processing by a 3D graphics system, the process including the step of generating at least one set mask image command including:
- a command identifier field including a six-bit binary value of 111110, and
 - a base address specifying the memory address of a top left corner of at least one depth image.
105. A system for providing at least one mask image mode command for processing by a 3D graphics system, the system including:
- at least one processor,
 - at least one memory, and
 - means coupled to the processor and to the memory for providing at least one set mask image command including:
- a command identifier field including a six-bit binary value of 111110, and

整理番号=0 0 0 5 6 8

(46)

a base address specifying the memory address of a top left corner of at least one depth image.

106. In a 3D graphics system, a process for interpreting at least one mask image mode command including:

interpreting a command identifier field including a six-bit binary value of 111110,

and

interpreting a base address specifying the memory address of a top left corner of at least one depth image.

107. In a 3D graphics system, a decoder for interpreting at least one mask image mode command including a six-bit binary value of 111110, the system including:

means for interpreting a command identifier field including a six-bit binary value of

111110, and

means for interpreting a base address specifying the memory address of a top left corner of at least one depth image.

108. A storage medium for use with a 3D graphics system, the storage medium for storing at least one mask image mode command including:

a command identifier field including a six-bit binary value of 111110, and

a base address specifying the memory address of a top left corner of at least one depth image.

109. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one triangle drawing command having a command identifier field including a six-bit binary value within the range of 001000 to 001111 followed by at least x and y position values, the triangle drawing command format specifying at least one triangle at the x and y positions corresponding to the x and y position values.

整理番号=000568

(47)

110. A system for generating at least one display command for processing by a 3D graphics system, the system including:
- at least one processor,
 - at least one memory, and
 - means coupled to the processor and to the memory for providing at least one triangle drawing command having a command identifier field including a six-bit binary value within the range of 001000 to 001111 followed by at least x and y position values, the triangle drawing command format specifying at least one triangle at the x and y positions corresponding to the x and y position values.
111. In a 3D graphics system, a process for executing at least one display command for processing including the steps of:
- (a) interpreting at least one triangle drawing command having a command identifier field including a six-bit binary value within the range of 001000 to 001111 followed by at least x and y position values, and
 - (b) rendering at least one primitive at the x and y positions corresponding to the x and y position values.
112. A 3D graphics system for executing at least one display command including a six-bit binary value within the range of 001000 to 001111 followed by at least x and y position values, the system including:
- a decoder that interprets at least one triangle drawing command having a command identifier field including a six-bit binary value within the range of 001000 to 001111 followed by at least x and y position values, and
 - a display processor that renders at least one primitive at the x and y positions corresponding to the x and y position values.
113. A storage medium for use with a 3D graphics system, the storage medium storing at least one display command having a command identifier field including a six-bit

整理番号=000568

(48)

binary value within the range of 001000 to 001111 followed by at least x and y position values, the triangle drawing command format specifying at least one triangle at the x and y positions corresponding to the x and y position values.

114. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one triangle drawing command including:

a command identifier field having a six-bit binary value of at least one of 001111

and 001011,

a set of edge coefficients,

a set of texture coefficients, and

a set of z buffer coefficients,

the triangle drawing command format specifying at least one triangle to be drawn in accordance with the set of edge coefficients, filled with a texture based at least in part on the set of texture coefficients, and z buffered based at least in part on the set of z buffer coefficients.

115. A system for providing at least one display command for processing by a 3D graphics system, the system including

at least one processor,

at least one memory, and

circuitry coupled to the processor and to the memory for providing at least one triangle drawing command including:

a command identifier field having a six-bit binary value of at least one of 001111

and 001011,

a set of edge coefficients,

a set of texture coefficients, and

a set of z buffer coefficients,

整理番号=000568

(49)

the triangle drawing command format specifying at least one triangle to be drawn in accordance with the set of edge coefficients, filled with a texture based at least in part on the set of texture coefficients, and z buffered based at least in part on the set of z buffer coefficients.

116. In a 3D graphics system, a process for executing at least one triangle drawing command including:
- (a) interpreting a command identifier field having a six-bit binary value of at least one of 001111 and 001011,
 - (b) interpreting a set of edge coefficients,
 - (c) interpreting a set of texture coefficients,
 - (c) interpreting a set of z buffer coefficients, and
 - (e) rendering at least one triangle in accordance with the set of edge coefficients, filled with a texture based at least in part on the set of texture coefficients, and z buffered based at least in part on the set of z buffer coefficients.
117. A 3D graphics system for executing at least one triangle drawing command having a command identifier field having a six-bit binary value of at least one of 001111 and 001011, the system including:
- (a) means for interpreting a command identifier field having a six-bit binary value of at least one of 001111 and 001011,
 - (b) means for interpreting a set of edge coefficients,
 - (c) means for interpreting a set of texture coefficients,
 - (c) means for interpreting a set of z buffer coefficients, and
- means coupled to the above-mentioned means for rendering at least one triangle in accordance with the set of edge coefficients, filled with a texture based at least in part on the set of texture coefficients, and z buffered based at least in part on the set of z buffer coefficients.

整理番号=0 0 0 5 6 8

(50)

118. A storage medium for use with a 3D graphics system, the storage medium for storing at least one triangle drawing command including:
- a command identifier field having a six-bit binary value of at least one of 001111 and 001011,
 - a set of edge coefficients,
 - a set of texture coefficients, and
 - a set of z buffer coefficients,
- the triangle drawing command format specifying at least one triangle to be drawn in accordance with the set of edge coefficients, filled with a texture based at least in part on the set of texture coefficients, and z buffered based at least in part on the set of z buffer coefficients.
119. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one triangle drawing command including:
- a command identifier field having a six-bit binary value selected from the set of 001010 and 001110,
 - a set of edge coefficients, and
 - a set of texture coefficients,
- the triangle drawing command format specifying at least one triangle to be drawn in accordance with the set of edge coefficients, and filled with a texture based at least in part on the set of texture coefficients.
120. A system for generating at least one display command for processing by a 3D graphics system, the system including:
- at least one processor,
 - at least one memory, and
 - circuitry coupled to the processor and to the memory for generating at least one triangle drawing command including:

整理番号=000568

(51)

a command identifier field having a six-bit binary value selected from the set of 001010 and 001110,

a set of edge coefficients, and

a set of texture coefficients,

the triangle drawing command format specifying at least one triangle to be drawn in accordance with the set of edge coefficients, and filled with a texture based at least in part on the set of texture coefficients.

121. In a 3D graphics system, a process for executing at least one triangle drawing command including:

(a) interpreting a command identifier field having a six-bit binary value selected from the set of 001010 and 001110,

(b) interpreting a set of edge coefficients,

(c) interpreting a set of texture coefficients, and

(d) drawing at least one triangle in accordance with the set of edge coefficients, and filled with a texture based at least in part on the set of texture coefficients.

122. A 3D graphics system for executing at least one triangle drawing command having a command identifier field having a six-bit binary value selected from the set of 001010 and 001110, the system including:

a command identifier field decoder that interprets a command identifier field having a six-bit binary value selected from the set of 001010 and 001110,

a rasterizer that interprets a set of edge coefficients,

a texture coordinate unit that interprets a set of texture coefficients, and

a display processor that draws at least one triangle in accordance with the set of edge coefficients, and filled with a texture based at least in part on the set of texture coefficients.

整理番号= 0 0 0 5 6 8

(52)

123. A storage medium for use with a 3D graphics system, the storage medium for storing at least one triangle drawing command including:
- a command identifier field having a six-bit binary value selected from the set of 001010 and 001110,
 - a set of edge coefficients, and
 - a set of texture coefficients,
- the triangle drawing command format specifying at least one triangle to be drawn in accordance with the set of edge coefficients, and filled with a texture based at least in part on the set of texture coefficients.
124. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one triangle drawing command including:
- a command identifier field having a six-bit binary value of a value selected from the set of 001001,
 - a set of edge coefficients, and
 - a set of z buffer coefficients,
- the triangle drawing command format specifying at least one non-shaded triangle to be drawn in accordance with the set of edge coefficients and z buffered based at least in part on the set of z buffer coefficients.
125. A system for providing at least one display command for processing by a 3D graphics system, the system including:
- at least one processor,
 - at least one memory, and
 - circuitry coupled to the processor and to the memory for providing at least one triangle drawing command including:
- a command identifier field having a six-bit binary value of a value selected from the set of 001001,

整理番号=000568

(53)

a set of edge coefficients, and

a set of z buffer coefficients,

the triangle drawing command format specifying at least one non-shaded triangle to be drawn in accordance with the set of edge coefficients and z buffered based at least in part on the set of z buffer coefficients.

126. In a 3D graphics system, a process for executing at least one triangle drawing command including:

(a) interpreting a command identifier field having a six-bit binary value of a value selected from the set of 001001,

(b) interpreting a set of edge coefficients,

(c) interpreting a set of z buffer coefficients, and

(d) drawing at least one non-shaded triangle in accordance with the set of edge coefficients and z buffered based at least in part on the set of z buffer coefficients.

127. A 3D graphics system for executing at least one triangle drawing command having a command identifier field having a six-bit binary value of a value selected from the set of 001001, the system including:

a decoder that interprets a command identifier field having a six-bit binary value of a value selected from the set of 001001,

an edge walker that interprets a set of edge coefficients,

a z buffer controller that interprets a set of z buffer coefficients, and

a display processor that draws at least one non-shaded triangle in accordance with the set of edge coefficients and z buffered based at least in part on the set of z buffer coefficients.

128. A storage medium for use with a 3D graphics system, the storage medium storing at least one triangle drawing command including:

整理番号=000568

(54)

a command identifier field having a six-bit binary value of a value selected from the set of 001001,

a set of edge coefficients, and

a set of z buffer coefficients,

the triangle drawing command format specifying at least one non-shaded triangle to be drawn in accordance with the set of edge coefficients and z buffered based at least in part on the set of z buffer coefficients.

129. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one textured rectangle drawing command including:

a command identifier field having a six-bit binary value within the range of 100100 and 100101,

at least two x coordinate values,

at least two y coordinate values,

a set of texture coefficients, and

a tile descriptor index value,

the textured rectangle drawing command format specifying at least one rectangle to be drawn in accordance with the x and y coordinate values, and filled with a texture based at least in part on the set of texture coefficients and the tile descriptor index value.

130. A system for providing at least one display command for processing by a 3D graphics system, the system including:

at least one processor,

at least one memory, and

means coupled to the processor and to the memory for providing at least one textured rectangle drawing command including:

a command identifier field having a six-bit binary value within the range of 100100 and 100101,

整理番号=000568

(55)

- at least two x coordinate values,
- at least two y coordinate values,
- a set of texture coefficients, and
- a tile descriptor index value,

the textured rectangle drawing command format specifying at least one rectangle to be drawn in accordance with the x and y coordinate values, and filled with a texture based at least in part on the set of texture coefficients and the tile descriptor index value.

131. In a 3D graphics system, a process for executing at least one textured rectangle drawing command including:
- (a) interpreting a command identifier field having a six-bit binary value within the range of 100100 and 100101,
 - (b) interpreting at least two x coordinate values,
 - (c) interpreting at least two y coordinate values,
 - (d) interpreting a set of texture coefficients,
 - (e) interpreting a tile descriptor index value, and
 - (f) rendering at least one rectangle in accordance with the x and y coordinate values, and filled with a texture based at least in part on the set of texture coefficients and the tile descriptor index value.
132. A 3D graphics system for executing at least one textured rectangle drawing command having a command identifier field having a six-bit binary value within the range of 100100 and 100101, the system including:
- a decoder that interprets a command identifier field having a six-bit binary value within the range of 100100 and 100101,
 - a processor that interprets at least two x coordinate values and at least two y coordinate values,
 - a texture coordinate unit that interprets a set of texture coefficients, and a tile descriptor index value, and

整理番号=000568

(56)

a display processor that renders at least one rectangle in accordance with the x and y coordinate values, and filled with a texture based at least in part on the set of texture coefficients and the tile descriptor index value.

133. A storage medium for use with a 3D graphics system, the storage medium for storing at least one textured rectangle drawing command including:
a command identifier field having a six-bit binary value within the range of 100100 and 100101,

- at least two x coordinate values,
- at least two y coordinate values,
- a set of texture coefficients, and
- a tile descriptor index value,

the textured rectangle drawing command format specifying at least one rectangle to be drawn in accordance with the x and y coordinate values, and filled with a texture based at least in part on the set of texture coefficients and the tile descriptor index value.

134. A process for generating at least one display command for processing by a 3D graphics system, the process including the steps of:
(a) generating at least one set primitive color command including:
a command identifier field having a six-bit binary value of 111010, and
a set of color coordinates; and

- (b) generating at least one filled rectangle drawing command including:
a command identifier field having a six-bit binary value of 110110,
at least two x coordinate values, and
at least two y coordinate values,

the filled rectangle drawing command format specifying at least one rectangle to be drawn in accordance with the x and y coordinate values, and filled with a color based at least in part on the color coordinates.

整理番号=000568

(57)

135. A system for generating at least one display command for processing by a 3D graphics system, the system including:
- at least one processor,
 - at least one memory, and
 - means coupled to the processor and to the memory for generating:
 - (a) at least one set primitive color command including:
 - a command identifier field having a six-bit binary value of 111010, and
 - a set of color coordinates; and
 - (b) at least one filled rectangle drawing command including:
 - a command identifier field having a six-bit binary value of 110110,
 - at least two x coordinate values, and
 - at least two y coordinate values,
- the filled rectangle drawing command format specifying at least one rectangle to be drawn in accordance with the x and y coordinate values, and filled with a color based at least in part on the color coordinates.
136. In a 3D graphics system, a process for executing at least one display command for processing by a 3D graphics system, the process including the steps of:
- (a) interpreting at least one set primitive color command including:
 - a command identifier field having a six-bit binary value of 111010, and
 - a set of color coordinates.;
 - (b) interpreting at least one filled rectangle drawing command including:
 - a command identifier field having a six-bit binary value of 110110,
 - at least two x coordinate values, and
 - at least two y coordinate values; and
 - (c) drawing at least one rectangle in accordance with the x and y coordinate values, and filled with a color based at least in part on the color coordinates.

整理番号= 0 0 0 5 6 8

(58)

137. In a 3D graphics system, an apparatus for executing at least one display command having a command identifier field having a six-bit binary value of 111010, the apparatus including the following structures:
- means for interpreting at least one set primitive color command including:
 - a command identifier field having a six-bit binary value of 111010, and
 - a set of color coordinates.;
 - means for interpreting at least one filled rectangle drawing command including:
 - a command identifier field having a six-bit binary value of 110110,
 - at least two x coordinate values, and
 - at least two y coordinate values; and
 - means for drawing at least one rectangle in accordance with the x and y coordinate values, and filled with a color based at least in part on the color coordinates.
138. A storage medium for use with a 3D graphics system, the storage medium for storing:
- (a) at least one set primitive color command including:
 - a command identifier field having a six-bit binary value of 111010, and
 - a set of color coordinates; and
 - (b) at least one filled rectangle drawing command including:
 - a command identifier field having a six-bit binary value of 110110,
 - at least two x coordinate values, and
 - at least two y coordinate values,
- the filled rectangle drawing command format specifying at least one rectangle to be drawn in accordance with the x and y coordinate values, and filled with a color based at least in part on the color coordinates.
139. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one texture defining command including:

整理番号=000568

(59)

a command identifier field having a six-bit binary value of a value of 110101,
an image data format parameter,
a color element size parameter,
a tile line size parameter,
a starting texture memory address,
a tile descriptor index,
a palette number,
at least one texture coordinate clamp enable parameter,
at least one texture coordinate mirror enable parameter,
at least one texture coordinate wrapping/mirroring mask, and
at least one texture coordinate level of detail shift parameter.

140. A system for providing at least one display command for processing by a 3D graphics system, the system including:
at least one processor,
at least one memory, and
means coupled to the processor and to the memory for
providing at least one texture defining command including:

a command identifier field having a six-bit binary value of a value of 110101,
an image data format parameter,
a color element size parameter,
a tile line size parameter,
a starting texture memory address,
a tile descriptor index,
a palette number,
at least one texture coordinate clamp enable parameter,
at least one texture coordinate mirror enable parameter,
at least one texture coordinate wrapping/mirroring mask, and
at least one texture coordinate level of detail shift parameter.

整理番号=000568

(60)

141. In a 3D graphics system, a process for processing at least one texture defining command including:

(a) interpreting a command identifier field having a six-bit binary value of a value of 110101,

(b) interpreting an image data format parameter,

(c) interpreting a color element size parameter,

(d) interpreting a tile line size parameter,

(e) interpreting a starting texture memory address,

(f) interpreting a tile descriptor index,

(g) interpreting a palette number,

(h) interpreting at least one texture coordinate clamp enable parameter,

(i) interpreting at least one texture coordinate mirror enable parameter,

(j) interpreting at least one texture coordinate wrapping/mirroring mask,

(k) interpreting at least one texture coordinate level of detail shift parameter, and

(l) generating at least one image based at least in part on the above-mentioned steps.

142. A 3D graphics apparatus for processing at least one texture defining command having a command identifier field having a six-bit binary value of a value of 110101, the apparatus including:

means for interpreting a command identifier field having a six-bit binary value of a value of 110101,

a circuit that interprets an image data format parameter, a color element size parameter, a tile line size parameter, a starting texture memory address, a tile descriptor index, a palette number, at least one texture coordinate clamp enable parameter, at least one texture coordinate mirror enable parameter, at least one texture coordinate wrapping/mirroring mask, and at least one texture coordinate level of detail shift parameter, and

整理番号=000568(61)

a display processor that generates at least one image based at least in part on the above-mentioned structures.

143. A storage medium for use with a 3D graphics system, the storage medium for storing at least one texture defining command including:
- a command identifier field having a six-bit binary value of a value of 110101,
 - an image data format parameter,
 - a color element size parameter,
 - a tile line size parameter,
 - a starting texture memory address,
 - a tile descriptor index,
 - a palette number,
 - at least one texture coordinate clamp enable parameter,
 - at least one texture coordinate mirror enable parameter,
 - at least one texture coordinate wrapping/mirroring mask, and
 - at least one texture coordinate level of detail shift parameter.
144. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one texture tile command including:
- a command identifier field having a six-bit binary value of a value within the set of 110100 and 110010,
 - low and high tile S coordinates,
 - low and high tile T coordinates, and
 - a tile descriptor index.
145. A system for providing at least one display command for processing by a 3D graphics system, the system including:
- at least one processor,

整理番号=000568

(62)

- at least one memory, and
circuitry coupled to the processor and to the memory for
providing at least one texture tile command including:
a command identifier field having a six-bit binary value of a value within the set of
110100 and 110010,
low and high tile S coordinates,
low and high tile T coordinates, and
a tile descriptor index.
146. In a 3D graphics system, a process for executing at least one texture tile command
including:
(a) interpreting a command identifier field having a six-bit binary value of a value
within the set of 110100 and 110010,
(b) interpreting low and high tile S coordinates,
(c) interpreting low and high tile T coordinates,
(d) interpreting a tile descriptor index, and
(e) generating a display based at least in part on steps (a)–(d).
147. A 3D graphics system for executing at least one texture tile command having a
command identifier field having a six-bit binary value of a value within the set of
110100 and 110010, the system including:
a decoder that interprets a command identifier field having a six-bit binary value of
a value within the set of 110100 and 110010,
a texture unit that interprets low and high tile S coordinates, low and high tile T
coordinates, a tile descriptor index; and
a display processor circuit that generates a display based at least in part on the
coordinates identifier and command identifier.

整理番号=000568

(63)

148. A storage medium for use with a 3D graphics system, the storage medium for storing at least one texture tile command including:
- a command identifier field having a six-bit binary value of a value within the set of 110100 and 110010,
 - low and high tile S coordinates,
 - low and high tile T coordinates, and
 - a tile descriptor index.
149. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one texture memory block loading command including:
- a command identifier field having a six-bit binary value of 110011,
 - low and high tile S coordinate parameters,
 - a low tile T coordinate parameter,
 - a T increment value, and
 - a tile descriptor index.
150. A system for generating at least one display command for processing by a 3D graphics system, the system including:
- at least one processor,
 - at least one memory, and
 - means coupled to the processor and to the memory for generating at least one texture memory block loading command including:
- a command identifier field having a six-bit binary value of 110011,
 - low and high tile S coordinate parameters,
 - a low tile T coordinate parameter,
 - a T increment value, and
 - a tile descriptor index.

整理番号=000568

(64)

151. In a 3D graphics system, a process for executing at least one texture memory block loading command including:
- (a) interpreting a command identifier field having a six-bit binary value of 110011,
 - (b) Interpreting low and high tile S coordinate parameters,
 - (c) interpreting a low tile T coordinate parameter,
 - (d) interpreting a T increment value,
 - (e) interpreting a tile descriptor index, and
 - (f) displaying at least one textured primitive based at least in part on steps (a)–(e).
152. A 3D graphics system for executing at least one texture memory block loading command having a command identifier field having a six-bit binary value of 110011, the system including:
- means for interpreting a command identifier field having a six-bit binary value of 110011,
- means for interpreting low and high tile S coordinate parameters,
- means for interpreting a low tile T coordinate parameter,
- means for interpreting a T increment value,
- means for interpreting a tile descriptor index, and
- a display circuit for displaying at least one textured primitive based at least in part on the S and T coordinate parameters, the T increment value, and the tile descriptor index.
153. A storage medium for use with a 3D graphics system, the storage medium for storing at least one texture memory block loading command including:
- a command identifier field having a six-bit binary value of 110011,
- low and high tile S coordinate parameters,
- a low tile T coordinate parameter,
- a T increment value, and
- a tile descriptor index.

整理番号=000568(65)

154. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one load texture look up table command including:
a command identifier field having a six-bit binary value of 110000,
low and high indices into the table, and
a tile descriptor index.
155. A system for providing at least one display command for processing by a 3D graphics system, the system including:
at least one processor,
at least one memory, and
circuitry coupled to the processor and to the memory for
providing at least one load texture look up table command including:
a command identifier field having a six-bit binary value of 110000,
low and high indices into the table, and
a tile descriptor index.
156. In a 3D graphics system, a process for executing at least one load texture look up table command including:
(a) interpreting a command identifier field having a six-bit binary value of 110000,
(b) interpreting low and high indices into the table,
(c) a tile descriptor index, and
(d) loading a texture look up table into a memory based at least in part on steps (a)–(c).
157. A 3D graphics system for executing at least one load texture look up table command having a command identifier field having a six-bit binary value of 110000, the system including:

整理番号= 0 0 0 5 6 8

(66)

a texture memory means for interpreting a command identifier field having a six-bit binary value of 110000,

means for interpreting low and high indices into the table,

means for interpreting a tile descriptor index, and

means for loading a texture look up table into the texture memory based at least in part on the command identifier field and the indices.

158. A storage medium for use with a 3D graphics system, the storage medium for storing at least one load texture look up table command including:
a command identifier field having a six-bit binary value of 110000,
low and high indices into the table, and
a tile descriptor index.
159. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one set color command including:
a command identifier field having a six-bit binary value within the range of 110111 to 111011,
a red component parameter,
a green component parameter,
a blue component parameter, and
an alpha component parameter.
160. A process as in claim 156 further including the step of generating an additional level of detail fraction field and an associated minimum clamp parameter.
161. A system for generating at least one display command for processing by a 3D graphics system, the system including:
at least one processor,

整理番号=000568

(67)

- at least one memory, and
circuitry coupled to the processor and to the memory for
generating at least one set color command including:
a command identifier field having a six-bit binary value within the range of 110111
to 111011,
a red component parameter,
a green component parameter,
a blue component parameter, and
an alpha component parameter.
162. A system as in claim 158 further including circuitry for providing an additional level
of detail fraction field and an associated minimum clamp parameter.
163. In a 3D graphics system, a process for interpreting at least one set color command
including:
(a) interpreting a command identifier field having a six-bit binary value within the
range of 110111 to 111011,
(b) interpreting a red component parameter,
(c) interpreting a green component parameter,
(d) interpreting a blue component parameter,
(e) interpreting an alpha component parameter, and
(f) generating an image based at least in part on steps (a)–(e).
164. A process as in claim 160 further including the step of interpreting an additional
level of detail fraction field and an associated minimum clamp parameter.
165. A 3D graphics system for interpreting at least one set color command having a
command identifier field having a six-bit binary value within the range of 110111 to
111011, the system including:

整理番号=000568

(68)

- means for interpreting a command identifier field having a six-bit binary value within the range of 110111 to 111011,
means for interpreting a red component parameter,
means for interpreting a green component parameter,
means for interpreting a blue component parameter,
means for interpreting an alpha component parameter, and
a display circuit that generates an image based at least in part on the parameters.
166. An apparatus as in claim 162 further including means for interpreting an additional level of detail fraction field and an associated minimum clamp parameter.
167. A storage medium for use with a 3D graphics system, the storage medium storing at least one set color command including:
a command identifier field having a six-bit binary value within the range of 110111 to 111011,
a red component parameter,
a green component parameter,
a blue component parameter, and
an alpha component parameter.
168. A storage medium as in claim 164 further including means for storing an additional level of detail fraction field and an associated minimum clamp parameter.
169. A process for generating at least one display command for processing by a 3D graphics system, the process including the step of generating at least one set primitive depth command including:
a command identifier field having a six-bit binary value of 101110,
a primitive depth parameter, and
a primitive delta depth parameter.

整理番号=000568

(69)

170. A system for providing at least one display command for processing by a 3D graphics system, the system including:
at least one processor,
at least one memory, and
means coupled to the processor and to the memory for
providing at least one set primitive depth command including:
a command identifier field having a six-bit binary value of 101110,
a primitive depth parameter, and
a primitive delta depth parameter.
171. In a 3D graphics system, a process for processing at least one set primitive depth command including:
(a) interpreting a command identifier field having a six-bit binary value of 101110,
(b) interpreting a primitive depth parameter,
(c) interpreting a primitive delta depth parameter, and
(d) generating at least one image based at least in part on steps (a)-(c).
172. A 3D graphics system for processing at least one set primitive depth command having a command identifier field having a six-bit binary value of 101110, the system including:
means for interpreting a command identifier field having a six-bit binary value of 101110,
means for interpreting a primitive depth parameter,
means for interpreting a primitive delta depth parameter, and
a display circuit that generates at least one image based at least in part on the parameters.
173. A storage medium for use with a 3D graphics system, the storage medium storing at least one set primitive depth command including:

整理番号=000568

(70)

a command identifier field having a six-bit binary value of 101110,
a primitive depth parameter, and
a primitive delta depth parameter.

3. Detailed Description of Invention

[FIELD OF THE INVENTION]

The present invention relates to low cost video game systems. More particularly, the invention relates to a video game system that can model a world in three dimensions and project the model onto a two dimensional viewing plane selected based on a changeable viewpoint.

[BACKGROUND AND SUMMARY OF THE INVENTION]

People's Imagination are fueled by visual images. What we actually see at sunset, what we dream at night, the pictures we paint in our mind when we read a novel--all of these memorable scenes are composed of visual images. Throughout history, people have tried to record these images with pencil or paints or video tape. But only with the advent of the computer can we begin to create with the same vividness, detail and realism that display in the real world or in the imagination.

Computer-based home video game machines such as the Nintendo Entertainment System and the Super Nintendo Entertainment System have been highly successful because they can interactively produce exciting video graphics. However, without additional add-on hardware, these prior video graphics systems generally operated in two dimensions, creating graphics displays from flat (planar) image representations in a manner somewhat analogous to tacking flat paper cutouts onto a bulletin board. Although very exciting game play can be created using two dimensional graphics techniques, a 2D system cannot provide the realism offered by three-dimensional graphics system.

整理番号=000568(71)

3D graphics are fundamentally different from 2D graphics. In 3D graphics techniques, a "world" is represented in three dimensional space. The system can allow the user to select a viewpoint within the world. The system creates an image by "projecting" the world based on the selected viewpoint. The result is a true three-dimensional image having depth and realism.

For many years, specialists have used super computers and high end workstations to create incredible realistic 3D images -- for example, ultra-detailed models of cars, planes and molecules; virtual reality as seen from the cockpit of a jet fighter or the front seat of an Olympic bobsled; and dinosaurs of "Jurassic Park." However, in the past, computer systems required to produce such images interactively cost tens of thousands of dollars -- well beyond the reach of the average consumer.

The low cost high performance 3D graphics system disclosed herein is intended to for the first time give millions of game players, not just the specialists, the chance to interact right inside these magnificent virtual 3D worlds with a richly featured high performance low cost system. What players get is truly amazing -- many times the power of any home computer system, far more realistic 3-dimensional animation, stunning graphics -- all delivered at a sufficiently low cost to be within the reach of the average consumer.

The following are a few examples of the many advantageous features provided by a system in accordance with the present invention:

- Realistic interactive 3D graphics in a low price system
- Optimum feature set/architecture for a low cost system for use with a color television set to provide video game play and other graphics applications in a low cost system and/or to produce particular screen effects
- Coprocessor that provides high performance 3D graphics and digital sound processing

整理番号=000568(72)

- Signal processor sharing between graphics digital processing and audio signal processing to achieve high quality stereo sound and 3-D graphics in a low cost color television based system
- Unified RAM approach increases flexibility
- All major system components can communicate through the shared RAM
- Techniques/structures for compensating for narrow main memory bus width
- Executable code from a storage device (e.g., a portable memory cartridge) can be loaded into the common RAM and accessed by the main processor through coprocessor memory access/arbitration circuitry
- Graphics coprocessor loadable microcode store receives microcode from a portable storage medium to provide additional flexibility and simplify compatibility issues
- Microcode is loaded via execution of "boot ROM" instructions
- Optimal commands and associated formats are used to invoke graphics and audio functions within the coprocessor and provide an interface between the graphics coprocessor and the rest of the system
- Coprocessor register set including particular hardware register definitions, formats and associated functions

整理番号=000568

(73)

- Microcode graphics and audio structure/processes provide efficient high performance operation
- Vector unit provides optimal performance for graphics and audio digital processing in a low cost package
- Pipelined rasterizing engine provides a one-pixel-per-cycle and two-pixel-per-cycle modes to minimize hardware cost while providing a rich feature set
- Low coprocessor pin out

[DETAILED DESCRIPTION OF A PRESENTLY PREFERRED EXAMPLE EMBODIMENT]

Figure 1 shows an example embodiment video game system 50 in accordance with the present invention(s). Video game system 50 in this example includes a main unit 52, a video game storage device 54, and handheld controllers 56 (or other user input devices). In this example, main unit 52 connects to a conventional home color television set 58. Television set 58 displays 3D video game images on its television screen 60 and reproduces stereo sound through its loud speakers 62.

In this example, the video game storage device 54 is in the form of a replaceable memory cartridge insertable into a slot 64 on a top surface 66 of main unit 52. Video game storage device 54 can comprise, for example, a plastic housing 68 encasing a read only memory (ROM) chip 76. The read only memory 76 contains video game software in this example. When the video game storage device 54 is inserted into main unit slot 64, cartridge electrical contacts 74 mate with corresponding "edge connector" electrical contacts within the main unit. This action electrically connects the storage device's read only memory 76 to the electronics within main unit 52.

整理番号=000568

(74)

"Read only memory" chip 76 stores software instructions and other information pertaining to a particular video game. The read only memory chip 76 in one storage device 54 may, for example, contain instructions and other information for an adventure game. The read only memory chip 76 in another storage device 54 may contain instructions and information to play a driving or car race game. The read only memory chip 76 of still another storage device 54 may contain Instructions and information for playing an educational game. To play one game as opposed to another, the user of video game system 50 simply plugs the appropriate storage device 54 into main unit slot 64—thereby connecting the storage device's read only memory chip 76 (and any other circuitry the storage device may contain) to the main unit 52. This enables the main unit 52 to access the information contained within read only memory 76, which information controls the main unit to play the appropriate video game by displaying Images and reproducing sound on color television set 58 as specified under control of the video game software in the read only memory.

To play a video game using video game system 50, the user first connects main unit 52 to his or her color television set 58 by hooking a cable 78 between the two. Main unit 52 produces both "video" signals and "audio" signals for controlling color television set 58. The "video" signals are what controls the images displayed on the television screen 60, and the "audio" signals are played back as sound through television loudspeakers 62. Depending on the type of color television set 58, it may be necessary to use an additional unit called an "RF modulator" in line between main unit 52 and color television set 58. An "RF modulator" (not shown) converts the video and audio outputs of main unit 52 into a broadcast type television signal that can be received and processed using the television set's internal "tuner."

The user also needs to connect main unit 52 to a power source. This power source may comprise a conventional AC adapter (not shown) that plugs into a standard home electrical wall socket and converts the house current into a lower DC voltage signal suitable for powering main unit 52.

整理番号=000568

(75)

The user may then connect hand controllers 56a, 56b to corresponding connectors 80 on main unit front panel 82. Controllers 56 may take a variety of forms. In this example, the controllers 56 shown each include various push buttons 84 and a directional switch or other control 86. The directional switch 88 can be used, for example, to specify the direction (up, down, left or right) that a character displayed on television screen 60 should move and/or to specify a point of view in a 3D world. Other possibilities include, for example, joysticks, mice pointer controls and other conventional user input devices. In this example, up to four controllers 56 can be connected to main unit 52 to allow 4-player games.

The user then selects a storage device 54 containing the video game he or she wants to play, and inserts that storage device into main unit slot 64 (thereby electrically connecting read only memory 76 to the main unit electronics via a printed circuit board 70 and associated edge contacts 74). The user may then operate a power switch 88 to turn on the video game system 50. This causes main unit 52 to begin playing the video game based on the software stored in read only memory 54. He or she may operate controllers 86 to provide inputs to main unit 52 and thus affect the video game play. For example, depressing one of push buttons 84 may cause the game to start. As mentioned before, moving directional switches 86 can cause animated characters to move on the television screen 60 in different directions or can change the user's point of view in a 3D world. Depending upon the particular video game stored within the storage device 54, these various controls 84, 86 on the controller 56 can perform different functions at different times. If the user wants to restart game play, he or she can press a reset button 90.

[EXAMPLE 3D SCREEN EFFECTS]

System 50 is capable of processing, interactively in real time, a digital representation or model of a three-dimensional world to display the world (or portions of it) from any arbitrary viewpoint within the world. For example, system 50 can interactively change the viewpoint in response to real time inputs from game controllers 86. This can permit, for example, the game player to see the world through the eyes of a "virtual

整理番号=0 0 0 5 6 8(76)

person" who moves through the world, and looks and goes wherever the game player commands him or her to go. This capability of displaying quality 3D images interactively in real time can create very realistic and exciting game play.

Figures 2(a)-3(f) show just one example of some three-dimensional screen effects that system 50 can generate on the screen of color television set 58. Figures 2(a)-3(f) are in black and white because patents cannot print in color, but system 50 can display these different screens in brilliant color on the color television set. Moreover, system 50 can create these images very rapidly (e.g., seconds or tenths of seconds) in real time response to operation of game controllers 86.

Each of Figures 2(a)-3(f) was generated using a three-dimensional model of a "world" that represents a castle on a hilltop. This model is made up of geometric shapes (i.e., polygons) and "textures" (digitally stored pictures) that are "mapped" onto the surfaces defined by the geometric shapes. System 50 sizes, rotates and moves these geometric shapes appropriately, "projects" them, and puts them all together to provide a realistic image of the three-dimensional world from any arbitrary viewpoint. System 50 can do this interactively in real time response to a person's operation of game controllers 86.

Figures 2(a)-2C and 3(f) show aerial views of the castle from four different viewpoints. Notice that each of the views is in perspective. System 50 can generate these views (and views in between) interactively in a matter of seconds with little or no discernible delay so it appears as if the video game player is actually flying over the castle.

Figures 3(d) and 3(e) show views from the ground looking up at or near the castle main gate. System 50 can generate these views interactively in real time response to game controller inputs commanding the viewpoint to "land" in front of the castle, and commanding the "virtual viewer" (i.e., the imaginary person moving through the 3-D world through whose eyes the scenes are displayed) to face in different directions. Figure 3(d) shows an example of "texture mapping" in which a texture (picture) of a brick wall is mapped onto the castle walls to create a very realistic image.

整理番号=000568(77)**[Overall Video Game System Electronics]**

Figure 4 shows that the principal electronics within main unit 52 includes a main processor 100, a coprocessor 200, and main memory 300. Main processor 100 is a computer that runs the video game program provided by storage device 54 based on inputs provided by controllers 56. Coprocessor 200 generates images and sound based on instructions and commands it gets from main processor 100. Main memory 300 is a fast memory that stores the information main processor 100 and coprocessor 200 need to work, and is shared between the main processor and the coprocessor. In this example, all accesses to main memory 300 are through coprocessor 200.

In this example, the main processor 100 accesses the video game program through coprocessor 200 over a communication path 102 between the main processor and the coprocessor 200. Main processor 100 can read from storage device 54 via another communication path 104 between the coprocessor and the video game storage device. The main processor 100 can copy the video game program from the video game storage device 54 into main memory 300 over path 106, and can then access the video game program in main memory 300 via coprocessor 200 and paths 102, 106.

Main processor 100 generates, from time to time, lists of commands that tell the coprocessor 200 what to do. Coprocessor 200 in this example comprises a special purpose high performance application-specific integrated circuit (ASIC) having an internal design that is optimized for rapidly processing 3-D graphics and digital audio. In response to commands provided by main processor 100 over path 102, coprocessor 200 generates video and audio for application to color television set 58. The coprocessor 200 uses graphics, audio and other data stored within main memory 300 and/or video game storage device 54 to generate images and sound.

Figure 4 shows that coprocessor 200 in this example includes a signal processor 400 and a display processor 500. Signal processor 400 is an embedded programmable microcontroller that performs graphics geometry processing and audio digital signal processing under control of a "microcode" computer program supplied by video game storage device 54. Display processor 500 is a high speed state machine that renders

整理番号=000568(78)

graphics primitives, thereby creating images for display on television 58. The signal processor 400 and display processor 500 work independently, but the signal processor can supervise the display processor by sending graphics commands to it. Both signal processor 400 and display processor 500 can be controlled directly by main processor 100. The following are examples of functions and operations the signal processor 400 and display processor 500 can perform:

SIGNAL PROCESSOR

- Matrix control
- 3D transformations
- Lighting
- Clipping, perspective and viewport application
- Display processor command generation

DISPLAY PROCESSOR

- Rasterization
- Texture coordinate generation
- Texture application and filtering
- Color combining
- Blending
- Fogging
- Antialiasing
- Frame buffer and frame buffer control

Figure 5 shows the main processes performed by the main processor 100, coprocessor 200 and main memory 300 in this example system 50. The main processor 100 receives inputs from the game controllers 56 and executes the video game program provided by storage device 54 to provide game processing (block 120). It provides animation, and assembles graphics and sound commands for use by coprocessor 200. The graphics and sound commands generated by main processor 100 are processed by

整理番号=000568(79)

blocks 122, 124 and 126—each of which is performed by coprocessor 200. In this example, the coprocessor signal processor 400 performs 3D geometry transformation and lighting processing (block 122) to generate graphics display commands for display processor 500. Display processor 500 “draws” graphics primitives (e.g., lines, triangles and rectangles) to create an image for display on color TV 58. Display processor 500 performs this “drawing” or rendering function by “rasterizing” each primitive and applying a texture to it if desired (block 126). It does this very rapidly—e.g., on the order of many millions of “pixels” (color television picture elements) a second. Display processor 500 writes its image output into a frame buffer in main memory 300 (block 128). This frame buffer stores a digital representation of the image to be displayed on the television screen 60. Additional circuitry within coprocessor 200 reads the information from the frame buffer and outputs it to television 58 for display (block 130).

Signal processor 400 also processes sound commands received from main processor 100 using digital audio signal processing techniques (block 124). Signal processor 400 writes its digital audio output into a sound buffer in main memory 300. The main memory temporarily “buffers” (i.e., stores) the sound output (block 132). Other circuitry in coprocessor 200 reads this buffered sound data from main memory 300 and converts it into electrical audio signals (stereo left and right channels) for application to and reproduction by television speakers 62a, 62b (block 134).

Television 58 displays 30 or 60 new images a second. This “frame rate” fools the human eye into seeing continuous motion, allowing main unit 52 to create animation effects on television screen 60 by changing the image slightly from one frame to the next. To keep up with this television frame rate, coprocessor 200 must create a new image every 1/30 or 1/60 of a second. Coprocessor 200 must also be able to produce a stream of continuous sound to go along with the animation effects on screen 60.

[Overall System Operation]

Figure 6 shows the overall operation of system 50 in more detail, and Figure 7 shows overall steps performed by the system to generate graphics. In this example, main

整理番号=000568(80)

processor 100 reads a video game program 108 stored in main memory 300 (generally, this video game program will have originated in video game storage device 54 and have been copied from the video game storage device into the main memory). In response to executing this video game program 108 (and in response to inputs from game controllers 56), main processor 100 creates (or reads from storage device 58) a list 110 of commands for coprocessor 200 (Figure 7, block 120a). This list 110, in general, includes two kinds of commands:

- (1) graphics commands
- (2) audio commands.

Graphics commands tell coprocessor 200 what images to generate on TV screen 60. Audio commands tell sound coprocessor 200 what sounds it should generate for reproduction on TV loudspeakers 62.

The list of graphics commands is called a "display list" because it controls the images coprocessor 200 displays on the TV screen 60. The list of audio commands is called a "play list" because it controls the sounds that are played over loudspeaker 62. Generally, main processor 100 specifies both a new display list and a new play list for each video "frame" time of color television set 58.

In this example, main processor 100 provides its display/play list 110 to coprocessor 200 by storing it into main memory 300 and then telling the coprocessor where to find it (Figure 7, block 120c). Main processor 100 also makes sure the main memory 300 contains a graphics and audio database 112 that includes all of the data coprocessor 200 will need to generate the graphics and sound requested in the display/play list 110. Some or all of this graphics and audio database 112 can come from storage device 54. The display/play list 110 specifies which portions of graphics and audio database 112 the coprocessor 200 should use. Main processor 100 also is responsible for making sure that signal processor 400 has loaded "microcode"—i.e., a computer program that tells the signal processor what to do.

Signal processor 400 reads the display/play list 110 from main memory 100 (Figure 7, block 122a) and processes this list—accessing additional data within the

整理番号=000568(81)

graphics and audio databas 112 as needed (Figure 7, block 122b). Signal processor 400 generates two main outputs: graphics display commands 112 for further processing by display processor 500 (Figure 7, block 122c); and audio output data 114 for temporary storage within main memory 300. Signal processor 400 processes the audio data in much less than the time it takes to play the audio through loudspeakers 62. Another part of the coprocessor 200 called an "audio interface" (not shown) subsequently reads the buffered audio data and outputs it in real time for reproduction by television loudspeakers 62.

The signal processor 400 can provide the graphics display commands 112 directly to display processor 500 over a path internal to coprocessor 200, or it may write those graphics display commands into main memory 300 for retrieval by the display processor (not shown). These graphics display commands 112 command display processor 500 to draw ("render") specified geometric shapes with specified characteristics (Figure 7, block 126a). For example, display processor 500 can draw lines, triangles or rectangles (polygons) based on these graphics display commands 112, and may fill triangles and rectangles with particular colors and/or textures 116 (e.g., images of leaves of a tree or bricks of a brick wall)—all as specified by the graphics display commands 112. Main processor 100 stores the texture images 116 into main memory 300 for access by display processor 500. It is also possible for main processor 100 to write graphics display commands 112 directly into main memory 300 for retrieval by display processor 500 to directly command the display processor.

Display processor 500 generates, as its output, a digitized representation of the image that is to appear on television screen 60 (Figure 7, block 126b). This digitized image, sometimes called a "bit map," is stored within a frame buffer 118 residing in main memory 300. Display processor 500 can also store and use a depth (Z) buffer 118b in main memory 300 to store depth information for the image. Another part of coprocessor 200 called the "video interface" (not shown) reads the frame buffer 118 and converts its contents into video signals for application to color television set 58 (Figure 7, block 127). Typically, frame buffer 118 is "double buffered," meaning that coprocessor 200 can be

整理番号= 0 0 0 5 6 8

(82)

writing the "next" image into half of the frame buffer while the video interface is reading out the other half.

The various steps shown in Figure 7 and described above are "pipelined" in this example. "Pipelining" means that different operations are performed concurrently for different stages in the graphics generation process. A simple analogy is the way most people do laundry. A non-pipeline mode of doing laundry would involve completing all relevant tasks (washing, drying, ironing/folding, and putting away) for one load of laundry before beginning the next load. To save time, people with multiple loads of laundry "pipeline" the laundry process by performing washing, drying, ironing/folding and putting away operations concurrently for different loads of laundry.

Similarly, the operations performed by main processor 100, signal processor 400, display processor 500 and video interface 210 are "pipelined" in this example. For example, main processor 100 in this example can be assembling a display list two video frames ahead while signal processor 400 and display processor 500 are processing data for one video frame ahead and video interface 210 is processing data for the current video frame in progress. As is explained below, the detailed graphics rendering steps performed by display processor 500 in block 126a are also pipelined to maximize speed performance.

[More Detailed System Architecture]

Figure 8 shows a more detailed architecture of video game system 50. This diagram shows video game main unit 52 including, in addition to main processor 100, coprocessor 200 and main memory 300, additional components such as a clock generator 136, a serial peripheral interface 138, an audio digital-to-analog converter (DAC) 140, an audio amplifier/mixer 142, a video digital-to-analog converter 144, and a video encoder 146.

In this example, the clock generator 136 (which may be controlled by a crystal 148) produces timing signals to time and synchronize the other components of main unit 52. Different main unit components require different clocking frequencies, and clock

整理番号=000568(83)

generator 136 provides suitable such clock frequency outputs (or frequencies from which suitable clock frequencies can be derived such as by dividing). A timing block 216 within coprocessor 200 receives clocking signals from clock generator 136 and distributes them (after appropriate dividing as necessary) to the various other circuits within the coprocessor.

In this example, the game controllers 58 are not connected directly to main processor 100, but instead are connected to main unit 52 through serial peripheral interface 138. Serial peripheral interface 138 demultiplexes serial data signals incoming from up to four (or five) game controllers 56 (or other serial peripheral devices) and provides this data in a predetermined format to main processor 100 via coprocessor 200. Serial peripheral interface 138 is bidirectional in this example, i.e., it is capable of transmitting serial information specified by main processor 100 in addition to receiving serial information.

Serial peripheral interface 138 in this example also includes a "boot ROM" read only memory 150 that stores a small amount of initial program load (IPL) code. This IPL code stored within boot ROM 150 is executed by main processor 100 at time of startup and/or reset to allow the main processor to begin executing game program instructions 108a within storage device 54 (see Figure 9, blocks 160a, 160b). The initial game program instructions 108a may, in turn, control main processor 100 to initialize the drivers and controllers it needs to access main memory 300 (see Figure 9, blocks 160c, 160d) and to copy the video game program and data into the faster main memory 300 for execution and use by main processor 100 and coprocessor 200 (see Figure 9, blocks 160e, 160f, 160g).

Also in this example, serial peripheral interface 138 includes a security processor (e.g., a small microprocessor) that communicates with an associated security processor 152 (e.g., another small microprocessor) within storage device 54 (see Figure 8). This pair of security processors (one in the storage device 54, the other in the main unit 52) perform an authentication function to ensure that only authorized storage devices may be used with video game main unit 52. See U.S. Patent No. 4,799,635. In this example, the

整理番号=0 0 0 5 6 8(84)

security processor within serial peripheral interface 138 may process data received from game controllers 56 under software control in addition to performing a security function under software control.

Figure 8 shows a connector 154 within video game main unit 52. This connector 154 connects to the electrical contacts 74 at the edge of storage device printed circuit board 70 in this example (see Figure 1). Thus, connector 154 electrically connects coprocessor 200 to storage device ROM 76. Additionally, connector 154 connects the storage device security processor 152 to the main unit's serial peripheral interface 138. Although connector 154 in the particular example is used primarily to read data and instructions from a non-writable read only memory 76, system 52 is designed so that the connector is bidirectional, i.e., the main unit can send information to the storage device 54 in addition to reading information from it.

Figure 8 also shows that the audio and video outputs of coprocessor 200 are processed by some electronics outside of the coprocessor before being sent to television set 58. In particular, in this example coprocessor 200 outputs its audio and video information in digital form, but conventional home color television sets 58 generally require analog audio and video signals. Therefore, the digital outputs of coprocessor 200 are converted into analog form—a function performed for the audio information by DAC 140 and for the video information by VDAC 144. The analog audio output of DAC 140 is amplified by an audio amplifier 142 that may also mix audio signals generated externally of main unit 52 and supplied through connector 154. The analog video output of VDAC 144 is provided to video encoder 146, which may, for example, convert "RGB" input signals to composite video outputs. The amplified stereo audio output of amplifier 142 and the composite video output of video encoder 146 are provided to home color television set 58 through a connector not shown.

As shown in Figure 8, main memory 300 stores the video game program in the form of CPU instructions 108b. These CPU instructions 108b are typically copied from storage device 54. Although CPU 100 in this example is capable of executing instructions directly out of storage device ROM 76, the amount of time required to access each

整理番号=000568(85)

instruction from the ROM is much greater than the time required to access instructions from main memory 300. Therefore, main processor 100 typically copies the game program/data 108a from ROM 76 into main memory 300 on an as-needed basis in blocks, and accesses the main memory in order to actually execute the instructions (see Figure 9, blocks 160e, 160f). The main processor 100 preferably includes an internal cache memory to further decrease instruction access time.

Figure 8 shows that storage device 54 also stores a database of graphics and sound data 112a needed to provide the graphics and sound of the particular video game. Main processor 100 reads the graphics and sound data 112a from storage device 54 on an as-needed basis and stores it into main memory 300 in the form of texture data 116, sound data 112b and graphics data 112c. In this example, display processor 500 includes an internal texture memory 502 into which the texture data 116 is copied on an as-needed basis for use by the display processor.

Storage device 54 also stores coprocessor microcode 156. As described above, in this example signal processor 400 executes a computer program to perform its various graphics and audio functions. This computer program or "microcode," is provided by storage device 54. Because the microcode 156 is provided by storage device 54, different storage devices can provide different microcodes—thereby tailoring the particular functions provided by coprocessor 200 under software control. Typically, main processor 100 copies a part of the microcode 156 into main memory 300 whenever it starts the signal processor, and the signal processor 400 then accesses other parts of the microcode on an as-needed basis. The signal processor 400 executes the microcode out of an instruction memory 402 within the signal processor 400. Because the SP microcode 156 may be too large to fit into the signal processor's internal instruction memory 402 all at once, different microcode portions may need to be loaded from main memory 300 into the instruction memory 402 to allow signal processor 400 to perform different tasks. For example, one part of the SP microcode 156 may be loaded into signal processor 400 for graphics processing, and another part of microcode may be loaded into the signal processor for audio processing. In this example, the signal processor microcode RAM

整理番号=0 0 0 5 6 8

(86)

402 (and an additional signal processor data memory RAM not shown in Figure 8) is mapped into the address space of main processor 100 so the main processor can directly access the RAM contents under software control through load and store instructions.

[Main Processor 100]

Main processor 100 in this example is a MIPS R4300 RISC microprocessor designed by MIPS Technologies, Inc., Mountain View, California. This R4300 processor includes an execution unit with a 64-bit register file for integer and floating-point operations, a 16 KB Instruction Cache, a 8 KB Write Back Data Cache, and a 32-entry TLB for virtual-to-physical address calculation. The main processor 100 executes CPU instructions (e.g., a video game program) 108 in kernel mode with 32-bit addresses. 64-bit integer operations are available in this mode, but 32-bit calling conventions are preferable to maximize performance. For more information on main processor 100, see, for example, Heinrich, MIPS Microprocessor R4000 User's Manual (MIPS Technologies, Inc., 1994, Second Ed.).

Main processor 100 communicates with coprocessor 200 over bus 102, which in this example comprises a bi-directional 32-bit SysAD multiplexed address/data bus, a bi-directional 5-bit wide SysCMD bus, and additional control and timing lines. See chapter 12 et seq. of the above-mentioned Heinrich manual.

The conventional R4300 main processor supports six hardware interrupts, one internal (timer) interrupt, two software interrupts, and one non-maskable interrupt (NMI). In this example, three of the six hardware interrupt inputs (INT0, INT1 and INT2) and the non-maskable interrupt (NMI) input allow other portions of system 50 to interrupt the main processor. Specifically, main processor INT0 is connected to allow coprocessor 200 to interrupt the main processor, main processor interrupt INT1 is connected to allow storage device 54 to interrupt the main processor, and main processor interrupts INT2 and NMI are connected to allow the serial peripheral interface 138 to interrupt the main processor. Any time the processor is interrupted, it looks at an internal interrupt register to determine the cause of the interrupt and then may respond in an appropriate manner (e.g., to read a

整理番号=000568

(87)

status register or perform other appropriate action). All but the NMI interrupt input from serial peripheral interface 138 are maskable (i.e., the main processor 100 can selectively enable and disable them under software control).

Main processor 100 reads data from and writes data to the rest of system 50 via the CPU-to-coprocessor bus 102. The coprocessor 200 performs a memory mapping function, allowing the main processor 100 to address main memory 300, the storage device cartridge ROM 76, the "boot ROM" 150 within serial peripheral interface 138 (and other parts of the serial peripheral interface), various parts of coprocessor 200 (including signal processor RAM 402), and other parts of system 50.

In the example, the operations performed by main processor 100 are completely dependent on videogame program 108. In this example, all "system" software is supplied by the storage device 58 to provide maximum flexibility. Different video games (or other applications) may run more efficiently with different kinds of high level software. Therefore, main unit 52 in this example does not provide any standard software libraries - or any software at all for that matter -- since such libraries could limit flexibility. Instead, all software in this example is supplied by storage device 54.

Developers of video game software 108 may wish to employ advanced software architecture such as, for example, device drivers, schedulers and thread libraries to manage the various resources within system 50. Since main processor 100 is a state-of-the-art RISC processor/computer, it is appropriate to use such software architecture/constructs and to implement video game program 108 in a high level software environment.

An example system "memory map" of the main processor 100 address space is shown in Figure 10. As shown in this Figure 10, main memory 300 is divided into two banks (bank 0 and bank 1) in this example. In addition, certain configuration registers 307 within the main memory 300 are mapped into the main processor address space, as are registers within coprocessor 200. Main processor 100 in this example can control each of the various coprocessor subblocks by writing, under control of video game program 108, into control registers associated with each coprocessor 200 sub-block.

整理番号=000568

(88)

As shown in Figure 10, storage device 54 address space is divided into two "domains" (for two different devices, for example). These "domains" are mapped into several parts of the main processor 100 address space. Various parts of the serial peripheral interface 138 (i.e., PIF boot ROM 150, a PIF buffer RAM, and a PIF status register) are also mapped into the main processor 100 address space.

[Unified Main Memory 300]

Main memory 300 in this example comprises a RDRAM dynamic random access memory available from Rambus Inc. of Mountain View, California. In this example, main memory 300 is expandable to provide up to 8 megabytes of storage, although main unit 52 may be shipped with less RAM (e.g., 2 or 3 MB) to decrease cost.

Main memory 300 provides storage for the entire system 50 in this example. It provides a single address space (see Figure 10 above) for storing all significant data structures, including for example (as shown in Figure 8):

- Main processor instructions 108
- Signal processor microcode 156
- Display list graphic commands 110a
- Play list audio commands 110b
- Texture maps 116 and other graphics data 112c
- Color image frame buffer 118a
- Depth (z) buffer 118b
- sound data 112b
- Audio output buffer 114
- Main processor working values
- Coprocessor working values
- Data communicated between various parts of the system.

Advantages and disadvantages in using single address space memory architectures for raster scan display systems are known (see, for example, Foley et al, Computer Graphics: Principles and Practice at 177-178 (2d Ed. Addison-Wesley 1990). Many video

整理番号=000568

(89)

game (and other graphics) system architects in the past rejected a single address space architecture in favor of using dedicated video RAM devices for graphics data and using other types of memory devices for other types of data. However, a unified main memory 300 provides a number of advantages in this particular example of a video game system 50. For example:

Data communications between system elements is simplified. Once data is stored in main memory 300, there is little or no additional overhead in communicating the data to another part of the system. The overhead of transferring data between different parts of the system is thus minimized. For example, since the main processor 100 and each sub-block within the coprocessor 200 can each access system main memory 300, the main memory used by all system elements for data structure storage can also be used as a general purpose communication channel/data buffer between elements.

For example, display lists 110 main processor 100 stores within main memory 300 can be directly accessed by signal processor 400. Similarly, display commands the main processor (and/or the signal processor) stores within the main memory can be directly accessed by display processor 500. The main processor 100 working data (which can automatically be written into the main memory 300 via a "cache flush") is immediately available to all other parts of the system.

The unified memory provides memory allocation flexibility. Main memory 300 locations look alike, and therefore each location can be used for storing any type of data structure. All main memory 300 allocation decisions are left to the application programmer. This provides great flexibility in terms of data structure sizes and memory usage. Data structures can be stored anywhere in main memory 300, and each location in memory 300 can be allocated however the application programmer specifies.

For example, one video game programmer might provide a large frame buffer for high resolution images and/or image scrolling and panning, while another programmer may decide to use a smaller frame buffer so as to free up memory space for other data structures (e.g., textures or audio data). One application may devote more of main memory 300 storage for audio data structures and less to graphics data, while another

整理番号=0 0 0 5 6 8(90)

application may allocate most of the storage for graphics related data. The same video game program 108 can dynamically shift memory allocation from one part of game play to another (e.g., at the time the game changes levels) to accomplish different effects. Application flexibility is not limited by any fixed or hardwired memory allocation.

The Unified RAM architecture supports flexible data structure sharing and usage. Since all significant data structures are stored within common main memory 300, they can all be accessed by main processor 100 and other system elements. There is no hardware distinction between display images and source images. For example, main processor 100 can, if desired, directly access individual pixels within frame buffer 118. The scan conversion output of display processor 500 can be used as a texture for a texture mapping process. Image source data and scan converted image data can be interchanged and/or combined to accomplish special effects such as, for example, warping scan-converted images into the viewpoint.

The shortcomings of a unified memory architecture (e.g., contention for access to the main memory 300 by different parts of the system) have been minimized through careful system design. Even though main memory 300 is accessed over a single narrow (9-bit-wide) bus 106 in this example, acceptable bandwidth has been provided by making the bus very fast (e.g., on the order of 240 MHz). Data caches are provided throughout the system 50 to make each sub-component more tolerant to waiting for main memory 300 to become available.

[Coprocessor 200]

Figure 8 shows that coprocessor 200 includes several components in addition to signal processor 400 and display processor 500, namely:

- CPU interface 202,
- a serial interface 204,
- a parallel peripheral interface 206,
- an audio interface 208,
- a video interface 210,

整理番号=000568

(91)

- a main memory DRAM controller/interface 212,
- a main internal bus 214 and
- a timing block 216.

In this example, main bus 214 allows each of the various main components within coprocessor 200 to communicate with one another.

Figure 11, a more detailed diagram of coprocessor 200, shows that the coprocessor is a collection of processors, memory interfaces and control logic all active at the same time and operating in parallel. The following briefly describes the overall functions provided by each of these other sub-blocks of coprocessor 200:

- Signal processor 400 is a microcoded engine that executes audio and graphics tasks.
- Display processor 500 is a graphics display pipeline that renders into frame buffer 118.
- Coprocessor serial interface 204 provides an interface between the serial peripheral interface 128 and coprocessor 200 in this example.
- Coprocessor parallel peripheral interface 206 interfaces with the storage device 54 or other parallel devices connected to connector 154.
- Audio interface 208 reads information from audio buffer 114 within main memory 300 and outputs it to audio DAC 140.
- Coprocessor video interface 210 reads information from frame buffer 118a within main memory 300 and outputs it to video DAC 144.
- The CPU interface 202 is the gateway between main processor 100, coprocessor 200 and the rest of system 50.
- DRAM controller/interface 212 is the gateway through which coprocessor 200 (and main processor 100) accesses main memory 300. Memory interface 212 provides access to main memory 300 for main processor 100, signal processor 400, display processor 500, video interface 210, audio interface 208, and serial and parallel interfaces 204, 206.

Each of these various processors and interfaces may be active at the same time.

Signal processor 400 in this example includes the instruction memory 402 discussed above, a data memory 404, a scalar processing unit 410 and a vector processing unit 420. Instruction memory 402 stores microcode for execution by scalar unit 410 and/or vector unit 420. Data memory 404 stores input data, work data and output data for the scalar unit 410 and for the vector unit 420. Signal processor 400 can execute instructions only out of instruction memory 402 in this example, but has access to main memory 300 via direct memory accessing (DMA) techniques.

In this example, scalar unit 410 is a general purpose Integer processor that executes a subset of the MIPS R4000 instruction set. It is used to perform general purpose operations specified by microcode within instruction memory 402. Vector unit 420 comprises eight 16-bit calculating elements capable of performing numerical calculations in parallel. Vector unit 420 is especially suited for graphics matrix calculations and certain kinds of digital audio signal processing operations.

Display processor 500 in this example is a graphics display pipelined engine that renders a digital representation of a display image. It operates based on graphics display commands generated by the signal processor 400 and/or main processor 100. Display processor 500 includes, in addition to texture memory 502, a rasterizer 504, a texture unit 506, a color combiner 508, a blender 510 and a memory interface 512. Briefly, rasterizer 504 rasterizes polygon (e.g., triangle, and rectangle) geometric primitives to determine which pixels on the display screen 60 are within these primitives. The texture unit can apply texture maps stored within texture memory 502 onto textured areas defined by primitive edge equations solved by rasterizer 504. The color combiner 508 combines and interpolates between the texture color and a color associated with the graphic primitive. Blender 510 blends the resulting pixels with pixels in frame buffer 118 (the pixels in the frame buffer are accessed via memory interface 512) and is also involved in performing Z buffering (i.e., for hidden surface removal and anti-aliasing operations). Memory interface 512 performs read, modify and write operations for the individual pixels, and also has special modes for loading/copying texture memory 502, filling rectangles (fast clears), and copying multiple pixels from the texture memory 502 into the frame buffer 118. Memory

interface 512 has one or more pixel caches to reduce the number of accesses to main memory 300.

Display processor 500 includes circuitry 514 that stores the state of the display processor. This state information is used by the rest of display processor 500 to, for example, select rendering modes and to ensure that all previous rendering effected by a mode change occurs before the mode change is implemented.

The command list for display processor 500 usually comes directly from signal processor 400 over a private "X bus" 218 that connects the signal processor to the display processor. More specifically, X-bus 218 in this example is used to transfer graphics display commands from the signal processor data memory 404 into a command buffer (not shown in Figure 11) within display processor 500 for processing by the display processor. However, in this example it is also possible for signal processor 400 and/or main processor 100 to feed graphics display commands to display processor 500 via main memory 300.

Display processor 500 accesses main memory 300 using physical addresses to load its internal texture memory 502, read frame buffer 118 for blending, read the Z buffer 118B for depth comparison, to write to the Z-buffer and the frame buffer, and to read any graphics display commands stored in the main memory.

[Coprocessor Internal Bus Architecture]

Figure 12 is a more detailed diagram showing an example coprocessor bus 214 arrangement, which in this example comprises a 32-bit address ("C") bus 214C and a 64-bit data ("D") bus 214D. These busses 214C, 214D are connected to each of signal processor 400, display processor 500, CPU interface 202, audio interface 208, video interface 210, serial interface 204, parallel peripheral interface 206, and main memory (RAM) interface 212. As shown in Figure 12, main processor 100 and each of the sub-blocks of coprocessor 200 communicates with main memory 300 via internal coprocessor busses 214C, 214D, and main memory interface/controller 212a/212b.

In this example, main memory interface/controller 212a, 212b converts main memory addresses asserted on coprocessor address bus 214C into 9-bit-wide format for communication over the 9-bit-wide main memory multiplexed address/data bus 106, and also converts between the main memory bus 106 9-bit-wide data format and the coprocessor data bus 214D 64-bit wide data format. In this example, the DRAM controller/interface 212 includes, as a part thereof, a conventional RAM controller 212b (see Figure 12) provided by Rambus Inc. The use of a 9-bit-wide main memory bus 106 reduces the chip pin count of coprocessor 200.

In this example, each of the coprocessor 200 sub-blocks shown has an associated direct memory access (DMA) circuit that allows it to independently address and access main memory 300. For example, signal processor DMA circuit 454, display processor DMA circuit 518, audio interface DMA circuit 1200, video interface DMA circuit 900, serial interface DMA circuit 1300, and parallel peripheral interface DMA circuit 1400 each allow their associated coprocessor sub-block to generate addresses on coprocessor address bus 214C and to communicate data via coprocessor data bus 214D (additionally, display processor 500 has a further memory interface block 512 for access to the main memory frame buffer 118 and texture data 116).

Although each of the coprocessor 200 sub-blocks can independently access main memory 300, they all share common busses 214C, 214D in this example -- and only one of the subblocks can use these shared busses at a time. Accordingly, coprocessor 200 has been designed to make most efficient use of the shared busses 214. For example, the coprocessor 200 sub-blocks may buffer or "cache" information to minimize the frequency of different bus accesses by the same sub-block and to make the subblocks more tolerant of temporary bus unavailability. A private bus 218 allows signal processor 400 to communicate with display processor 500 without having to wait for main bus 214 to become available.

Also as shown in Figure 12, each of the sub-blocks of coprocessor 200 includes control/status registers that can be accessed by main processor 100 via CPU interface 202. For example, signal processor registers 407, display processor registers 507, audio

整理番号=000568(95)

Interface registers 1207, video interface registers 907, serial interface registers 1307, parallel peripheral interface registers 206, RAM interface registers 1007a, and RAM controller registers 1007b are each mapped into the main processor 100 address space. The main processor 100 can read from and/or write to these various registers under control of game program 108 to directly control the operation of sub-blocks within coprocessor 200.

[Signal Processor 400]

Figure 13 shows the architecture of signal processor 400 of this example in more detail. As explained above, signal processor 400 includes a scalar unit 410, a vector unit 420, an instruction memory 402 and a data memory 404. In this example, scalar unit 410 is a 32-bit integer processor that executes a sub-set of the MIPS 4000 instruction set. Vector unit 420 (which is defined as a "CP1" coprocessor of scalar unit 410 under the MIPS 4000 architecture) performs integer calculations (e.g., multiplications, additions, subtractions and multiply/accumulates) on eight 16-bit sets of values in parallel.

Vector unit 420 can perform the same operation on eight pairs of 16-bit operands in parallel simultaneously. This makes signal processor 400 especially suited for "sum of products" calculations such as those found in matrix multiplications, texture resampling, and audio digital signal processing such as, for example, digital audio synthesis and spatial and frequency filtering.

Signal processor 400 uses a RISC (reduced instruction set computer) architecture to provide high performance machine control based on instructions residing in the instruction memory 402. In this example, execution unit includes a program counter 432 that is used to address instruction memory 402 over path 434. This program counter 432 can access only the 4 kilobyte instruction space within instruction memory 402 in this example--requiring that all instructions to be executed by the signal processor first be placed into the instruction memory. Execution unit 430 generates output control signals 436 based on the particular instructions currently being executed. These output control signals 436 control all other parts of signal processor 400, and are sequenced to manage

整理番号=0 0 0 5 6 8

(96)

pipelined instruction processing. Scalar unit 410 and vector unit 420 are controlled by these control signals 436. For example, scalar unit 410 may address data memory 404 via path 438 to read data from and/or write data into the data memory using load/store block 440. Data path 414 may perform tests based on results of calculations and provide resulting condition outputs to execution unit 430 via path 442. This execution unit 430 may use these condition outputs to perform a conditional branch or jump, loading program counter 432 with the appropriate (next) address into instruction memory 402. Because scalar processor 410 has these more general capabilities, it is used in this example for general purpose functions such as, for example, control flow, address calculation and the like--in addition to providing 32-bit integer calculations.

Execution unit 430 executes intermediate, jump and register instruction formats in accordance with the standard MIPS R4000 instruction set. Figure 14(a) shows an example of a register instruction format 450 and how signal processor 400 uses that register instruction format to access three 128-bit wide words 452 within data memory 404. Register instruction format 450 may include a 6-bit operation code field 450(a), a 5-bit source register specifier 450(b), a 5-bit target (source/destination) register specifier 450(c), a 5-bit destination register specifier 450(d), and a parameter field 450(e). The parameter field 450(e) may specify shift amounts and/or functions, and together with operation code 450(a) defines the operation to be performed. Each of fields 450(b), 450(c) and 450(d) specifies a location within data memory 404--and thus each designates 128-bit word.

As shown in Figure 14(b), vector unit 420 treats each of these 128-bit words as a concatenated sequence of eight 16-bit values, and operates on each of the 16-bit values in parallel. The operations of vector unit 420 are invoked by instructions within the CP1 type instructions typically reserved for floating point operations in the MIPS R4000 instruction set (signal processor 400 has no floating point unit in this example).

Scalar unit 410 includes a register file 412 comprising 32 registers, each register being 32 bits wide. Scalar unit also includes a data path 414 comprising adders, shifters, and other logic required to execute integer calculations and other operations. Register

整理番号=000568

(97)

file 412 is similar to the general purpose register file defined by the MIPS R4000 architecture, and accepts instructions in R4000 format. Data path 414 includes an integer multiplier/divider, and operates in conjunction with an execution unit 430 that receives 64-bit wide instructions from instruction memory 402.

Vector unit 420 includes eight sets of register files 422(0)-422(7) and eight sets of corresponding data paths 423 (0)-423(7). Data paths 423 each include a 16-bit multiplier, a 16-bit adder and a 48-bit accumulator (48 bit accumulation accommodates audio filters with a large number of taps, and also accommodates partial products wherein a series of 16-bit multiplies and sums is used to obtain a 32-bit result for certain graphics calculations requiring more than 16-bit precision). Each of register files 422 comprises 32 registers each of which are 32-bits wide. A 128 bit wide data path 444 connects vector unit 420 to load/store block 440, and another 128 bit wide data path 446 connects the load/store block 440 to data memory 404. Data memory 404 stores 4096 (4KB) words, each word being 128 bits wide. When a word in data memory 404 is retrieved for use by vector unit 420, it is sliced into eight 16-bit segments, with each segment being sent to a different register file 422 within vector unit 420 (see Figure 14(b)). Figure 14(c) shows an example add operation performed by vector unit 420. When vector unit 420 writes to a destination addressed within data memory 404, each of register files 422 contributes 16-bits which are combined into a 128 bit word before being written into the data memory (see Figure 14(a)). Alternatively, load/store block 440 includes a steering multiplexer arrangement (not shown) that can steer 16-bit sub-words within the data memory 128-bit word to/from different vector unit register files 422 -- with the particular sub-word and the particular vector unit register file being selectable based on instructions from instruction memory 402. Similarly, load/store block 440 includes a further steering multiplexer arrangement (not shown) that can steer different sized data units (e.g., bytes, 16-bit half-words, or 32-bit words) between data memory 408 and scalar unit 410 -- with the particular data unit and size being specified by instructions within instruction memory 402. See, for example, description of Load and Store "Byte", "Halfword", "Word", "Word Left" and "Word Right" in Heinrich, MIPS R4000 Microprocessor User's Manual (2d Ed. 1994).

Signal processor 400 also includes a DMA controller 454 and CPU control registers 456. DMA controller 454 is connected to the coprocessor internal bus 214, and is used to transfer data into and out of instruction memory 402 and/or data memory 404. For example, DMA controller 454 can copy microcode modules 156 from main memory 300 into signal processor instruction memory 402. DMA controller 454 may also be used to transfer information between data memory 404 and main memory 300. DMA controller 454 can be commanded by execution unit 430, and receives DMA address and data information from scalar unit data path 414 over path 438. DMA controller 454 may also be commanded by main processor 100 via CPU control registers 456. CPU control registers 456 are mapped into the main processor 100 address space, and can be accessed by signal processor 400 and execution unit 430 using MIPS "CP0" instruction formats.

Figures 15(d)-16(l) show example CPU control registers 756. The registers shown in Figures 15(d)-15(h) are used to control and/or monitor the DMA controller 454.

For example, the SP-DRAM DMA address register 458 shown in Figure 15(d) can be written to or read from by main processor 100 (as well as SP execution unit 430), and is used to specify a starting DMA address within instruction memory 402 or data memory 404. SP memory DMA address 460 shown in Figure 15(e) is used to specify a starting DMA address in main memory 300. Read and write DMA length registers 462, 464 shown in Figure 15(f) and 15(g), respectively, specify the length of a block of data to be transferred between signal processor 400 and main memory 300—with the direction of transfer depending upon which one of these two registers is used to specify the block length. DMA status registers 466, 468 shown in Figures 15(h) and 15(i) respectively, can be read by main processor 100 to determine whether DMA controller 454 is full or busy, respectively.

Figure 16(j) shows the main SP status register 470 within CPU control registers 456. SP status register 470 acts as an SP control register when it is written to by main processor 100 (top diagram of Figure 16(j)), and indicates SP status when read by the main processor (bottom diagram in Figure 16(j)). When used as a status register, SP

整理番号= 0 0 0 5 6 8

(99)

status register 470 tells main processor 100 whether the SP is halted (field 471), whether the SP is operating in a breakpoint mode (field 472), whether the DMA controller 454 is busy (field 474) or full (field 475), whether SP I/O is full (field 476), whether the SP is operating in single step mode (field 477), whether the SP is operating in a mode in which it won't generate an interrupt upon reaching a breakpoint (block 478), and whether the SP has generated various general purpose "signals" 479 that can be defined under software control to provide status concerning various software-dependent parameters. Main processor 100 can write to register 470 to stop or start signal processor 400 (fields 480, 481), to clear breakpoint mode (field 482), to clear or set an interrupt mode (fields 483, 484), to clear or set single step mode (fields 485, 486), to clear or set an interrupt on breakpoint mode (fields 487, 488), and to clear or set the various software-dependent "signals" (fields 489, 490).

Figure 16(k) shows an additional SP register 491 used as a "semaphore" for general purpose communications between the main processor 100 and the signal processor 400. This register 491 contains a flag that main processor 100 sets upon reading the register and clears upon writing to the register. Signal processor 400 can also set or clear this flag.

Figure 16(l) shows an SP instruction memory BIST status register 492 that is used as a BIST control register when written to by main processor 100 (top diagram in Figure 16(l)) and indicates BIST status when read by the main processor (bottom diagram of Figure 16(l)). Program counter 432 is preferably also mapped into the CPU control registers 456 so that it can be written to and read from by main processor 100.

[Signal Processor Microcode]

The particular functions signal processor 400 performs depend on the SP microcode 156 provided by storage device 54. In this example, SP microcode 156 provides both graphics and audio processing functions. As explained above, the main tasks performed by signal processor 400 for graphics processing include reading a display list, performing 3-dimensional geometry transformation and lighting calculations,

整理番号=000568(100)

and generating corresponding graphics display commands for use by display processor 500. In more detail, signal processor 400 performs the following overall graphics functions under control of microcode 156:

- Display list processing
- Matrix definition
- Vertex generation and lighting
- Texture definition/loading
- Clipping and culling
- Display processor command setup
- Flow control

Signal processor 400 performs the following overall functions under control of microcode 156 to process audio:

- Play list processing
- Digital audio synthesis/processing
- Writing digital audio samples to main memory audio buffer 114

[Task Lists]

Main processor 100 tells signal processor 400 what to do by providing the signal processor with a task list. The microcode 156 program that runs on signal processor 400 is called a task. Main processor 100 (and thus the video game program 108 supplied by storage device 54) is responsible for scheduling and invoking tasks on signal processor 400. The task list contains all of the information signal processor 400 needs to begin task execution, including pointers to the microcode 156 routines it needs to run in order to perform tasks. Main processor 100 provides this task list under control by game program 108.

Figure 17 shows an example of a task list 250. The task list 250 may reference one or more display lists and/or play lists 110. These display lists or play lists 110, in turn, may reference additional data structures including other display lists or play lists. A

整理番号=000568

(101)

display list 110 can point to other display lists and/or graphics data. Similarly, a play list can reference other play list and/or sound data. In this example, display lists and play lists can be thought of as hierarchical data structures up to ten levels deep. Signal processor 400 processes the display lists and play lists of the stack, pushing and popping the current display list pointer. All display lists must terminate with an "end" command. For example, display list 110(1) shown in Figure 17 references another display list 110(2). Display list 110(2) references graphics data 112 needed to execute the list. Similarly, play list 110(4) shown in Figure 17 references sound data 112B.

For graphics animation, it is desirable to "double buffer" only parts of the display list 110 that change from one frame to another. In this way, only the data that changes from one frame to the next needs to be "double buffered"—thus conserving space in main memory 300. Swapping between double buffers is efficiently done by changing segment base addresses within task lists 250 and by organizing the hierarchical display lists in an appropriately efficient manner. Display lists or fragments of display lists can be chained together for more efficient memory utilization.

Figure 18 shows an example process performed by main processor 100 to invoke processing of a new task list by signal processor 400. Main processor 100 first loads the task (display) list into main memory 300 (block 601). It then halts signal processor 400 (or checks to insure that the signal processor is halted) by writing to and/or reading from SP status register 470 (block 602). Main processor 100 then writes to SP DMA registers 458, 460, 462 to load an initial microcode module into signal processor instruction memory 402 (604, Figure 18). Main processor 100 next stores the address in main memory 300 of the task (display) list loaded by block 601 into signal processor data memory 404 (block 606, Figure 18). Main processor 100 then resets the signal processor program counter 432 (block 608, Figure 18), and writes to SP status register 470 to start the signal processor 400 (block 610, Figure 18). The signal processor 400 typically then uses its DMA controller 454 to fetch the task (display) list from main memory 300 into its data memory 404.

整理番号=0 0 0 5 6 8

(102)

Now that signal processor 400 has a task list and is started, it proceeds to perform each of the operations requested in the task list. It continues to execute the task list until it reaches the end of the task list, at which time it stops and waits for main processor 100 to provide a new task list. Generally, main processor 100 provides a new task list once each video frame--although, as discussed above, in many cases only a portion of the task list and/or the display and/or play lists the task list references may actually change from one frame to the next. Portions of the task list in main memory 300 may be "double buffered" so the main processor 100 can be writing to one buffer while signal processor 400 reads from another buffer. Before the next video frame, the main processor 100 can change a pointer to give the signal processor 400 access to the new buffer.

As signal processor 400 executes the task list, it retrieves additional SP microcode 156 modules from main memory 300 as needed to perform the specified tasks. For example, signal processor 400 may use its DMA facility 454 to load particular graphics microcode into instruction memory 402 to execute graphics commands specified by a task list, and may similarly retrieve and load audio processing microcode routines to perform audio processing specified by the task list. Different microcode routines or "overlays" may be loaded on an as-needed basis to more optimally handle particular types of graphics and/or audio processing operations. As one example, the signal processor 400 may load special lighting graphics routines as overlays to perform particular lighting operations, and may load clipping routines or overlays to perform particular culling operations. Microcode loading and reloading into signal processor 400 during execution of the single task list 250 is necessary in this example because signal processor instruction memory 402 is not large enough to store all of SP microcode 156, and the signal processor is designed so that it can execute instructions only out of its internal instruction memory.

Figure 19 shows an example of a simplified graphics process performed by signal processor 400 based on a display list 110. In this simplified process, the display list 110 first commands signal processor 400 to set various attributes defining the overall graphical images that are to be rendered by the co-processor. Such attributes include, for

整理番号=0 0 0 5 6 8

(103)

example, shading, lighting, Z buffering, texture generation, fogging and culling (Figure 19, block 612). The display list next commands signal processor 400 to define a modeling/viewing matrix and a projection matrix (Figure 19, block 614). Once the appropriate matrices have been defined, the display list commands signal processor 400 to transform a set of vertices based on the modeling/viewing matrix and the projection matrix defined by block 614 and also based on the attributes set by block 612 (Figure 19, block 616). Finally, the display list commands signal processor 400 to generate a graphics display (e.g., triangle) command that directs display processor 500 to render a primitive based on the vertices generated by block 616 and the attributes set by block 612 (Figure 19, block 618). Signal processor 400 may, in response to step 618, transfer the display processor command it has generated (or the address of the command, which the signal processor may store in its data memory 404 or in main memory 300) for access and execution by display processor 500.

Figure 20 shows an overall process 620 performed by signal processor graphics microcode 156 to process a display list 110 (e.g., to perform the type of process shown in Figure 19). Signal processor 400 gets the next display list command and determines what kind of a command it is (Figure 20, block 622). Display lists commands in this example generally have five different types:

- Signal processor attribute command
- Display processor command
- Matrix command
- Vertex command
- Triangle command
- Flow control command

If the display list command is a signal processor attribute command, signal processor 400 sets signal processor attributes as specified by the command (Figure 20, block 624). In this example, the following types of SP attribute command are defined:

- shading
- lighting

整理番号=0 0 0 5 6 8

(104)

- Z-buffering
- texturing
- fogging
- culling.

The following are example SP attribute command formats and associated definitions:

[SIGNAL PROCESSOR ATTRIBUTE COMMANDS]

G_SETGEOMETRYMODE:

command	
	command

This command "sets" some of the rendering pipeline state. This state is maintained in the signal processor 400, and a SET/CLEAR interface is presented to the user.

Bits which are "on" in the command field are turned ON in the internal state.

G_SHADE Enable vertex shading or use primitive color to paint the polygon (default is vertex shading).

G_LIGHTING Enable lighting calculations.

G_SHADING_SMOOTH Enable smooth or flat shading (the default, with this bit cleared is flat shading).

G_ZBUFFER Enable z-buffer depth calculations.

G_TEXTURE_GEN Enable automatic generation of the texture coordinates S & T. After transformations, a spherical mapping will be used to replace any S

整理番号=0 0 0 5 6 8

(105)

& T value originally given with the vertex.

G_FOG Enable fog coefficient to be generated and replace the vertex alpha. Large alphas are more foggy (farther).

G_TEXTURE_GEN_LINEAR Enable linearization of the texture coordinates generated when G_TEXTURE_GEN is set. For example, this allows the use of a panoramic texture map when performing environment mapping.

G_LOD Enable generation level of detail (LOD) value for mipmapped textures and texture-edge mode.

G_CULL_FRONT Cull the front-facing polygons.

G_CULL_BACK Cull the back-facing polygons.

G_CLEARGEOMETRY MODE:

Same as G_SETGEOMETRYMODE, but this command "clears" some of the rendering pipeline state (bits which are "on" in the command field are turned OFF in the internal state).

G_LIGHT:

Command		param	length = 16
	seg	address	

整理番号=000568

(106)

↓

light.r	light.g	light.b	0x00
light.r	light.g	light.b	0x00
light.x	light.y	light.z	0x00

This command passes a light to the rendering pipeline. There can be up to 7 directional lights (numbered 1-7) plus an ambient light. The param specifies which light number (n) to replace with this light description. Use the G_NUM_LIGHTS command to specify how many of the 8 lights to use. If the number of lights specified is N, then the first N lights (1-N) will be the ones used, and the Nth+1 lights will be the ambient light. The "param" field should be set based on a value maintained in data memory $404 + (n-1) \times 2$.

The ambient light is defined by a color: light.r, light.g, light.b (unsigned 8 bit integers) which should be set to the color of the ambient light multiplied by the color of the object which is to be drawn (If you are lighting a texture mapped object just use the color of the ambient light). (For ambient lights the light.x, light.y, and light.z fields are ignored). The ambient light cannot be turned off except by specifying a color of black in this example.

Directional lights are specified by a color: light.r, light.g, light.b (unsigned 8 bit integers) which, like the ambient light color, should be set to the color of the light source multiplied times the color of the object which is to be drawn. Directional lights also have a direction. The light.x, light.y, light.z fields (signed 8 bit fractions with 7 bits of fraction) indicates the direction from the object to light. There must be at least one directional light (if G_LIGHTING is enabled in G_SETGEOMETRYMODE command) turned on, but if its color is black it will have no effect on the scene.

The G_NUM_LIGHTS command should always be used sometime after G_LIGHT command(s) before the next G_VTX command even if the number of lights has not changed.

整理番号=000568

(107)

G_NUM_LIGHTS:

Command		param	length=8
	seg	address	

↓

0x8000	32 x (1+N)
0x00000000	

N = number of diffuse light sources (1-7).

This command specifies how many lights should be used. It should always be used after the G_LIGHT command before the next G_VTX command. The parameter specifies the number of diffuse light sources (N) which must be at least 1 and not more than 7. The ambient light source will be light number N+1 and the directional light sources will be lights numbered 1 through N.

G_SETOTHERMODE_H:

command		shift	len
word			

This command sets the high word of the "other" modes in the display processor, including blending, texturing, and frame buffer parameters. The signal processor 400 remembers the high and low words of the display processor 500 "other" state, in order to present a simple set-command interface. Although this is a display processor command, it must be parsed and interpreted by the signal processor 400 and therefore cannot be sent directly to the display processor without first going through the signal processor.

The shift and len parameters in this command are used to construct a mask:

整理番号=000568

(108)

$$(((0x01 \ll \text{len}) - 1) \ll \text{shift})$$

This mask is used to clear those bits in the display processor 500 status word. New bits, from the word parameter are OR'd into the status word. (the parameter word must be pre-shifted).

G_SETOTHERMODE_L

Same as G_SETOTHERMODE_H, but affects the low word of the "other" modes on the display processor 500.

G_TEXTURE:

command	s scale		
t scale	mipmap level	tile num	on

This command turns texture mapping ON/OFF, provides texture coordinate scaling, and selects the tile number (within a tiled texture). Scale parameters are in the format of (.16) and scale the texture parameters in vertex commands. Texture on/off turns on and off the texture coordinate processing in the geometry pipeline. Tile number corresponds to tiles chosen in the raster portion of the pipeline. The tile num also holds the maximum levels for level of detail (LOD) (mid-mapping).

整理番号=000568

(109)

G_LOOKAT_X:

Command		param	length=16
	seg	address	

↓

0x00000000			
0x00000000			
X	Y	Z	0x00

This command is used for automatic texture coordinate generation. It is used to describe the orientation of the eye so that the signal processor 400 knows with respect to what to generate texture coordinates. The XYZ values (8 bit signed fractions with 7 bits of fraction) describe a vector in worldspace (the space between the MODELVIEW matrix and the PROJECTION matrix) which is perpendicular to the viewer's viewing direction and pointing towards the viewer's right.

G_LOOKAT_Y:

Same as G_LOOKAT_X, but the first zero words in the addressed segment are zero (0x00000000).

[DP Command Generation]

Referring back to Figure 20, if the next display list command is one intended for display processor 500, signal processor 400 simply writes the command to the display processor (block 626 of Figure 20). Block 626 can either DMA the display processor command into display processor 500 via the X-bus 218, or it can deposit the display

整理番号=000568(110)

processor command in a buffer within main memory 300 for access by the display processor.

[MATRIX COMMANDS]

If the next display list command is a matrix command, signal processor 400 updates the state of the current matrix it is using (Figure 20, block 628) and places the updated matrix on the matrix stack (block 630). As mentioned above, in this example signal processor 400 maintains a 10-deep modeling/viewing matrix stack. New matrices can be loaded onto the stack, multiplied (concatenated) with the top of the stack, or popped off of the stack. In this example, signal processor 400 maintains a "one-deep" projection matrix. Therefore, new matrices can be loaded onto or multiplied with the current projection matrix, but cannot be pushed or popped.

In this example, the modeling/viewing matrix stack resides in main memory 300. The video game program 108 must allocate enough memory for this stack and provide a pointer to the stack area in task list 250. The format of the matrix is optimized for the signal processor's vector unit 420. To provide adequate resolution, signal processor 400 in this example represents each matrix value in 32-bit "double precision"—with an upper 16 bit signed integer portion (indicating the part of the value greater than 1) and a lower 16-bit fractional portion (indicating the part of the value between 0 and 1). However, vector unit 420 in this example operates on 16-bit wide values and cannot directly multiply 32-bit wide values. The matrix format (which is shown in Figure 21(b)) groups all of the integer parts of the elements, followed by all of the fractional parts of the elements. It allows signal processor 400 to more efficiently manipulate the matrix by multiplying 16 bit

整理番号=0 0 0 5 6 8

(111)

integer parts and 16 bit fractional parts separately without have to repeatedly "unpack" or "pack" the matrix.

For example, vector unit 420 can multiply each of the 16-bit fixed point signed integer values in a matrix row in one operation, it can multiply each of the 16-bit fractional portions of the same row in another operation. These two partial results can be added together to obtain a 32-bit double precision value, or they can be used separately (e.g., for operations that require only the integer part of the result or only the fractional part of the result). Thus, matrix representations thus allows signal processor 400 to efficiently process 32-bit precision values even though vector unit 420 in this example, operates on 16-bit values and as no explicit "double precision" capability.

The following are example signal processor matrix commands and associated formats:

[Example Matrix Commands]

G_MTX:

Command		param	length
	seg	address	

↓

m00 int	m00 frac
m10 int	m10 frac
...	...

The matrix command points to a 4x4 transformation matrix (See Figure 21(b)) that will be used to transform the subsequent geometry, in a manner controlled by the flags in the

整理番号=000568

(112)

parameter field. The length is the size of the incoming matrix in bytes. A 4x4 matrix pointed to by this command has the following format: It is a contiguous block of memory, containing the 16 elements of the matrix in ROW MAJOR order. Each element of the matrix is in a fixed point format, S15.16. The length of a 4 x 4 matrix in bytes should be 64 bytes. The segment id and address field are used to construct the main memory 300 address of the actual matrix. (see G_SEGMENT SP command for more information).

The following flags in the parameter field are used:

- | | |
|------------------|---|
| G_MTX_MODELVIEW | Identifies the incoming matrix as a modelview matrix, which is necessary to provide efficient transformation of polygon normals for shading, etc. (default) |
| G_MTX_PROJECTION | Identifies the incoming matrix as a projection matrix, which does not affect the transformation of the polygon normals for shading, etc. |
| G_MTX_MUL | The incoming matrix is concatenated with the current top of the matrix stack. (default) |
| G_MTX_LOAD | The incoming matrix replaces the current top of the (modelview or projection) matrix stack. |
| G_MTX_NOPUSH | The current top of the matrix stack is not pushed prior to performing the load or concat operation with the top of the stack. (default) |

整理番号=000568

(113)

G_MTX_PUSH

The current top of the matrix stack is pushed prior to performing the load or concat operation with the top of the stack. Push is only supported with G_MTX_MODELVIEW, and not with G_MTX_PROJECTION.--Since there is no projection matrix stack (the projection must be explicitly reloaded)

This single command with the combination of parameters allows for a variety of commonly used matrix operations. For example, (G_MTX_LOAD | G_MTX_NOPUSH) replaces the top of the stack. (G_MTX_MUL | G_MTX_PUSH) performs a concatenation while pushing the stack for typical modeling hierarchy construction.

For lighting and texturing, the polygon normal also must be transformed by the inverse transpose of the modelview matrix (reference the "OpenGL Programming Guide"). This is the reason separate modelview and projection stacks are kept, and incoming matrices must be identified.

G_POPMTX:

command	
	param

This command pops the modelview matrix stack. The parameter field should be 0. Popping an empty stack results in...(doesn't pop). Since there is no projection matrix stack, this command is supported only for the modelview matrix.

G_VIEWPORT:

整理番号=000568

(114)

Command		param	length=16
	seg	address	

↓

x scale	y scale
z scale	pad
x translate	y translate
z translate	pad

This command sends a viewport structure to the graphics pipeline.

The segment id and address field are used to construct the main memory 300 address of the actual VIEWPORT structure (see G_SEGMENT for more information).

The viewport transformation is a scale-translation of the normalized screen coordinates. In general, the viewport must be constructed in cooperation with the projection matrix in order to meet the hardware requirements for screen device coordinates.

The scale and translation terms for x and y have 2 bits of fraction, necessary to accommodate the sub-pixel positioning in the hardware. The z values have no fraction.

Accounting for the fractional bits, using one of the default projection matrices, the viewport structure can be initialized like this:

```
(SCREEN_WD/2*4, (SCREEN_HT/2) * 4, G_MAXZ, 0, /* scale */
```

```
(SCREEN_WD/2*4, (SCREEN_HT/2) * 4, 0, 0, /* translate */
```

[Vertex Command Processing]

整理番号=000568

(115)

Referring once again to Figure 20, if the next display list command is a "vert x command", signal processor 400 transforms the vertices specified by the vertex command by the current matrix state and possibly shaded by the current lighting state, performs a clip test on the vertices, and loads the resulting vertices into a vertex buffer 408 within data memory 404. Signal processor 400 in this example has a vertex buffer that holds up to sixteen vertices. Figure 22(a) shows the signal processor 400 vertex buffer, which is fully exposed to main processor 100 and thus to video game program 108. This internal vertex buffer 404, which can hold up to 16 points, is stored in signal processor data memory 404 and can be read by main processor 100.

Although signal processor 400 in this example, can handle only lines, triangles or rectangles (i.e., surfaces defined by 2, 3, or 4 vertices), vertex buffer 408 in this example, stores up to 16 vertices so that the signal processor can re-use transformed vertex values instead of having to recalculate the vertices each time. 3D authoring/modeling software used to create video game program 108, in this example, should preferably organize display list 110 to maximize vertex re-use (and thus speed performance).

Figure 22(b) shows an example vertex data structure signal processor 400 uses to represent each of the vertices stored in vertex buffer 408. In this example, the transformed x, y, z, and w, values corresponding to the vertex are stored in double precision format, with the integer parts first followed by the fractional parts (fields 408(1)(a)-408(1)(h)). With vertex color (r, g, b, α) are stored in fields 408(1)(i)-408(1)(l), and vertex texture coordinates (s, t) are stored in fields 408(1)(m), 408(1)(n). Additionally, from this example, the vertex values in screen space coordinates (i.e., transformed and projected onto the viewing plane) are stored in fields 408(1)(o)-408(1)(t) (with the one/w value stored in double precision format). The screen coordinates are used by display

整理番号=000568

(116)

processor 500 to draw polygons defined by the vertex. The transformed 3-dimensional coordinates are maintained in vertex buffer 408 for a clipping test. Since polygons (not vertices) are clipped, and since the vertices in vertex buffer 408 may be re-used for multiple polygons, these transformed 3D vertex values are stored for multiple possible clipping test to be performed. In addition, the vertex data structure 408(1) includes flags 408(1)(v) that signal processor 400 can use, for example, to specify clip test results (i.e., whether the vertex falls inside or outside of each of six different clip planes). The perspective projection factor stored in fields 408(1)(s), 408(1)(t) is retained for perspective correction operations performed by the display processor texture coordinate unit (explain below).

The following is an example of a vertex command format used to load the internal vertex buffer with some points:

G_VTX:

Command		n	vO	length
	seg	address		

↓

x		y	
z		flag	
s		t	
r or nx	g or ny	b or nz	a

•
•
•

整理番号=000568

(117)

This command loads (n+1) points into the vector buffer beginning at location v0 in the vertex buffer. The segment id and address field are used to construct the main memory 300 address of the actual VTX structure. (see G_SEGMENT for more information). The number of vertices n, is encoded as "the number minus one", in order to allow a full 16 vertices to be represented in 4 bits. The length is the number of points times 16, the size of the VTX structure (in bytes). Vertex coordinates are 16-bit integers, the texture coordinates s and t are S10.5. The flag parameter is ignored in this example. A vertex either has a color or a normal (for shading). Colors are 8 bit unsigned numbers. Normals are 8 bit signed fractions (7 bits of fraction). (0x7f maps to +1.0, 0x81 maps to -1.0, and 0x0 maps to 0.0). Normal vectors must be normalized, i.e.,

$$\sqrt{(x^2+y^2+z^2)} \leq 127$$

Upon receiving a vertex command, signal processor 400 transforms the vertices specified in the vertex command using the current modeling/viewing matrix (Figure 20, block 632). See Neider et al, Open GL Programming Guide (Silicon Graphics 1993) at chapter 3 ("viewing"). These transformations orient the object represented by the vertices in 3-dimensional space relative to the selected view point. For example, they may translate, rotate and/or scale the represented object relative to a selected point of view. Such transformation calculations make heavy use of the signal processor vector unit 420 and its ability to perform eight parallel calculations simultaneously. The transformed results are stored in vertex data structure fields 408(1)(a)–408(1)(h) in double precision format in this example.

[Clip Test]

整理番号=000568(118)

Signal processor 400 then performs a clip test (Figure 20, block 636) to determine whether the transformed vertex is inside or outside of the scene. Six clipping planes define the sides and ends of the viewing volume. Each transformed vertex is compared to each of these six planes, and the results of the comparison (i.e., on which side of the clip plane the vertex is located) are stored in vertex buffer "flags" field 408(v) (see Figure 22(b)). These results are used by clipping block 646 in response to a "triangle command" (see below). Note that because this example clips polygons and not vertices, Figure 20 block 636 does not actually perform clipping, it simply tests vertex position relative to the clip planes.

[Projection]

Signal processor 400 then transforms the vertex values using the projection matrix (Figure 20, block 638). The purpose of the projection transformation is to define a viewing volume, which is used in two ways. The viewing volume determines how an object is projected onto the 2-dimensional viewing screen (that is, by using a perspective or an orthographic projection). (See Open GL Programming Guide at 90 *et seq.*) The resulting transformed vertices have now been projected from 3-dimensional space onto the 2-dimensional viewing plane with the proper for shortening (if the projection matrix defines a perspective projection) or orthographically (if the projection matrix defines an orthographic projection). These screen coordinates values are also written to the vertex buffer data structure at fields 408(1)(o)–408(1)(t) (the "1/w" value is retained for later perspective correction).

[Lighting]

整理番号=000568(119)

Signal processor 400 next performs lighting calculations in order to "light" each of the vertices specified in the vertex command. System 50 supports a number of sophisticated real-time lighting effects, including ambient (uniform) lighting, diffuse (directional) lights, and specular highlights (using texture mapping). In order to perform lighting calculations in this example, signal processor 400 must first load an SP microcode 108 overlay to perform the lighting calculations. The `G_SETGEOMETRYMODE` command must have specified that lighting calculations are enabled, and the lights must have been defined by the `G_NUM_LIGHTS` command discussed above. The part of microcode 108 that performs the lighting calculations is not normally resident within signal processor 400, but is brought in through an overlay when lighting calls are made. This has performance implications for rendering scenes with some objects lighted and others colored statically. In this example, the lighting overlay overwrites the clipping microcode, so to achieve highest performance it is best to minimize or completely avoid clipped objects in lighted scenes.

To light an object, the vertices which make up the objects must have normals instead of colors specified. In this example, the normal consists of three signed 8-bit numbers representing the x, y and z components of the normal (see the `G_VTX` command format described above). Each component ranges in value from -128 to +127 in this example. The x component goes in the position of the red color of the vertex, the y into the green and the z into the blue. Alpha remains unchanged. The normal vector must be normalized, as discussed above.

Lighting can help achieve the effect of depth by altering the way objects appear as they change their orientation. Signal processor 400 in this example supports up to seven diffused lights in a scene. Each light has a direction and a color. Regardless of the

整理番号=0 0 0 5 6 8

(120)

orientation of the object and the viewer, each light will continue to shine in the same direction (relative to the open "world") until the light direction is changed. In addition, one ambient light provides uniform illumination. Shadows are not explicitly supported by signal processor 400 in this example.

As explained above, lighting information is passed to signal processor 400 in light data structures. The number of diffuse lights can vary from 0 to 7. Variables with red, green and blue values represent the color of the light and take on values ranging from 0 to 255. The variables with the x, y, z suffixes represent the direction of the light. The convention is that the direction points toward the light. This means the light direction indicates the direction to the light and not the direction that the light is shining (for example, if the light is coming from the upper left of the world the direction might be $x = -141, y = -141, z = 0$). To avoid any ambient light, the programmer must specify the ambient light is black (0, 0, 0.).

The G_light command is used to activate a set of lights on a display list. Once lights are activated, they remain on until the next set of lights is activated. This implies that setting up a new structure of lights overwrites the old structure of lights in signal processor 400. To turn on the lighting computation so that the lights can take effect, the lighting mode bit needs to be turned on using the G_SETGEOMETRYMODE command.

The lighting structures discussed above are used to provide color values for storing into vertex buffer fields 408(1)(i)-408(1)(l).

[Texture Coordinate Scaling/Creation]

Signal processor 400 next performs texture coordinate scaling and/or creation (Figure 20, block 642). In this example, the operations performed by block 642 may be

整理番号=000568(121)

used to accomplish specular highlighting, reflection mapping and environment mapping.

To render these effects, coprocessor 200 in this example uses a texture map of an image of the light or environment, and computes the texture coordinates s,t based on the angle from the viewpoint to the surface normal. This texture mapping technique avoids the need to calculate surface normals at each pixel to accomplish specular lighting. It would be too computationally intensive for system 50 in this example to perform such surface normal calculations at each pixel.

The specular highlight from most lights can be represented by a texture map defining a round dot with an exponential or Gaussian function representing the intensity distribution. If the scene contains highlights from other, oddly shaped lights such as fluorescent tubes or glowing swords, the difficulty in rendering is no greater provided a texture map of the highlight can be obtained.

Although display processor 500 performs texture mapping operations in this example, signal processor 400 performs texture coordinate transformations for each vertex when these effects are required. Activation or de-activation of the signal processor texture coordinate transformations is specified by a value within the G_SETGEOMETRYMODE Command (see above). In addition, the G_SETGEOMETRYMODE Command can specify linearization of the generated textured coordinates, e.g., to allow use of a panoramic texture map when performing environment mapping.

In this example, signal processor 400 texture coordinate generation utilizes the projection of the vertex normals in the x and y directions in screen space to derive the s and t indices respectively for referencing the texture. The angle between the viewpoint and the surface normal at each vertex is used to generate s, t. The normal projections

整理番号=000568(122)

are scaled to obtain the actual s and t values in this example. Signal processor 400 may map the vertices "behind" the point of view into 0, and may map positive projections into a scaled value.

In this example, texturing is activated using the G_TEXTURE command described above in the signal processor attribute command section. This command provides, among other things, scaling values for performing the texture coordinate mapping described above.

As explained above, the texture coordinate mapping performed by signal processor 400, in this example, also requires information specifying the orientation of the eye so that the angle between the vertex surface normal and the eye can be computed. The G_LOOKAT_X and the G_LOOKAT_Y commands supply the eye orientation for automatic texture coordinate generation performed by signal processor 400. The transformed texture coordinate values, if they are calculated, are stored by signal processor 400 in the vertex data structure at fields 408(1)(m), 408(1)(n). These texture coordinate values are provided to display processor 500 to perform acquired texture mapping using a texture specified by the G_TEXTURE command.

Since these effects use texture mapping, they cannot be used with objects which are otherwise texture mapped. .

[Vertex Buffer Write]

After performing all of these various steps, signal processor 400 writes the transformed, lighted, projected vertex values into vertex buffer 408 (Figure 20, block 644), and returns to parse the next display list command (block 622).

整理番号=000568

(123)

[Triangle Command Processing]

Once signal processor 400 has written vertices into its vertex buffer 408, the display list 110 can provide a "triangle command". The "triangle command," which specifies a polygon defined by vertices in vertex buffer 408, is essentially a request for signal processor 400 to generate a graphics display command representing a polygon and to send that command to display processor 500 for rendering. In this example, signal processor 400 can render three different kinds of primitives: lines, triangles and rectangles. Different modules of microcode 108 need to be loaded in this example to render lines or triangles. In this example, all rectangles are 2-dimensional primitives specified in screen-coordinates, and are neither clipped nor scissored.

The following is an example of a format and associated function of triangle commands:

Example of Triangle Commands

The following command specifies a triangle defined by 3 vertices in the vertex buffer:

G_TRI1:

command				
	N	v0	v1	v2

This command results in one triangle, using the vertices v0, v1, and v2 stored in the internal vertex buffer. The N field identifies which of the three vertices contains the normal of the face (for flat shading) or the color of the face (for flat shading).

The following command is used to control signal processor 400 to generate display processor 500 commands for rendering a line defined by two vertices in vertex buffer 408:

整理番号=000568

(124)

G_LINE3D:

command			
	N	v0	v1

This command generates one line, using the vertices v0 and v1 in the internal vertex buffer. The N field specifies which of the two vertices contain the color of the face (for flat shading).

Textured and filled rectangles require intervention by signal processor 400 and are thus a signal processor operation. The following is an example command format and associated function of a texture rectangle command:

G_TEXRECT

command	x0	y0
	x1	y1

command	0x000000	
S (top left texture coord)		T (top left texture coord)

command	0x000000	
DsDx		DtDy

整理番号= 0 0 0 5 6 8

(125)

These 3 commands draw a 2D rectangle with the current texture. The parameters x_0 , y_0 specify the upper left corner of the rectangle; x_1 , y_1 are the lower right corners. All coordinates are 12 bits. S and T are signed 10.5 bit numbers, and specify the upper left coordinate of s , t . $DsDx$ and $DtDy$ are signed 5.10 bit numbers, and specify change in s (t) per change in x (y) coordinate.

Signal processor 400 also in this example supports a `G_TEXRECT_FLIP` command that is identical to the `G_TEXRECT` command except that the texture is flipped so that the s coordinate changes in the y direction and the t coordinate changes in the x direction.

`G_FILLRECT`:

command	x_0	y_0
	x_1	y_1

This command draws a 2D rectangle in the current fill color. The parameters x_0 , y_0 specify the upper left corner of the rectangle; x_1 , y_1 are the lower right corners. All coordinates are 12 bits.

[Clipping/Setup]

Referring back to Figure 20, upon receipt of a triangle command, signal processor 400 performs any necessary clipping of the vertices (Figure 20, block 646). This clipping operation eliminates portions of geometric primitives that lie outside of the six clipped planes defining the view plane.

As explained above, the results of the clip test 636 performed for each vertex are stored and available in vertex buffer 408. With the triangle command now defining a

整理番号=000568

(126)

primitive defined by those vertices, signal processor 400 can proceed to clip the primitive.

If all of the vertices of a primitive lay within the space defined by the six clip planes, the entire primitive exists within the display space and does not need to be clipped. If all of the vertices defining a primitive lay outside of the same clip plane (as indicated by the flags field of vertex data structure 408(1) shown in Figure 22(b)), the entire primitive can be excluded from display and thus discarded. If some of the vertices defining a primitive lie within the display space and some lay outside of it (or if all vertices lay outside of the display space but define a primitive which passes through the displayed space), the primitive needs to be clipped and new vertices defined. These tests and operations are performed by clipping block 646 in this example.

Signal processor 400 next performs backface culling (Figure 20, block 647). This operation maximizes drawing speed by discarding polygons that can be determined to be on the backface of an object and thus hidden from view. In this example, either front-facing, back-facing, neither or both types of primitives can be culled (i.e., discarded) by block 647. The types of primitives to cull are specified by parameters in the G_SETGEOMETRYMODE command described above--allowing geometry to be ordered in any direction or where used with different culling flags to achieve various effects (e.g., interior surfaces, two-sided polygons, etc.).

Signal processor 400 also performs some set up operations (Figure 20, block 648), and may then pass a graphics display command to display processor 500 to control the display processor to render the primitive (Figure 20, block 650). As part of the set up operation (block 648), signal processor 400 in this example translates "segmented" addresses in the display list 110 into physical addresses that the display processor 500 can use (the display processor is a physical address machine in this example).

整理番号=000568

(127)

In this example, signal processor 400 uses a segment table 416 (see Figure 23) to assist it in addressing main memory 300. More specifically, addresses within signal processor 400 may be represented by a table entry 417A and a 26-bit offset 417B. The table entry 417A references one of 16 base addresses within segment address table 416. The referenced base address may be added to the offset 417b to generate a physical address into main memory 300. Signal processor 400 constructs a main memory 300 address by adding the base address for the segment and a 26-bit offset (which could be provided, for example, by a display list 110). The segment table 416 is constructed based on the following example G_SEGMENT command:

G_SEGMENT

command		
	seg	address

This command adds an entry in the segment table 416 discussed above.

The segmented addressing used by signal processor 400 in this example can be useful to facilitate double-buffered animation. For example, video game program 108 can keep two copies of certain display list fragments within main memory 300, with the same offsets in two different segments. Switching copies of them is as easy as swapping the segment pointers in signal processor 400. Another use is to group data and textures in one segment and to group static background geometry in another segment. Grouping data might help optimize memory caching in main processor 100. All data which contains embedded addresses must be preceded by the appropriate G_SEGMENT command that loads the signal processor 400 segment table with the proper base address.

整理番号=000568

(128)

Although signal processor 400 can use the segment addressing scheme shown in Figure 23, this arrangement is not available to display processor 500 in this example. Hence, part of set up processing 648 is to translate any segment addresses that point to data structures required for rendering into physical addresses that can be used directly by display processor 500.

[DP Command Write]

The primary output of signal processor 400 for graphics purposes is one or more commands to display processor 500 that are outputted by Figure 20, block 650. Although main processor 100 (or storage device 54) can directly supply display processor 500 commands, for 3D images the signal processor 400 generally needs to perform the transformation processes described above to generate display processor commands representing transformed, projected lighted, clipped, culled primitives.

The repertoire of display processor commands are shown in Figures 65-102. Signal processor 400 is responsible for formatting appropriately the display processor commands it generates, and for including the appropriate information and address information in the commands. In addition, signal processor 400 may generate and provide certain appropriate mode and attribute commands the display processor may require to render a particular primitive specified by the signal processor using the appropriate parameters (although many of the mode and attribute commands for the display processor 500 are typically supplied directly by main processor 100 under control of game program 108). As mentioned above, main processor 100 can provide any display processor 500 directly, but in general, needs to rely on the signal processor to

整理番号=000568(129)

generate at least some display processor commands whenever 3D objects need to be transformed.

[Flow Control Command Processing]

Referring once again to Figure 20, if the display list command received by signal processor 400 is a flow control command, then signal processor 400 will respond to this command in an appropriate manner to navigate through or traverse the display list 110. The following example commands and formats provide flow control.

整理番号=0 0 0 5 6 8

(130)

[Example Flow Control Commands]**G_DL:**

Command		param	(not used)
	seg	address	

↓

-
-
-

This command points to another display list and is used to create display list hierarchies, nested display lists, indirect references, etc. The segment field identifies a memory segment. The address field is the offset from the base of that segment. Together, these form an address in main memory 300 pointing to the new display list. A length field (not shown) may describe the length of the new display list in bytes—although in this example it is preferred that all display lists are terminated by a G_ENDDL command. The parameter field holds flags which control the behavior of the transfer. If

整理番号=0 0 0 5 6 8

(131)

the flag G_DL_NOPUSH is set, the current display list is not pushed onto the stack before transferring control. This behaves more like a branch or go to, rather than a hierarchical display list (this may be useful to break up a larger display list into non-contiguous memory pieces, then just connect them with display list branches).

G_ENDDL:

command	

The end display list command terminates this branch of the display list hierarchy, causing a "pop" in the processing of the display list hierarchy. This command is most useful for constructing display list pieces of variable or unknown size, terminated with an end command instead of providing a display list length a priori. All display lists must terminate with this command.

G_NOOP:

command			

This command does nothing. It is generated internally under some circumstances.

Figure 20, block 652 performs the function of maintaining a display list stack in main memory 300 and, pushing and nooping (traversing) this display list stack. Block 652 halts signal processor 400 when the signal processor encounters an "open end" display list command.

整理番号=000568

(132)

[SIGNAL PROCESSOR MICROCODE AUDIO PROCESSING]

Signal processor 400 in this example performs digital audio processing in addition to the graphics processing discussed above. Signal processor vector unit 420 is especially suited for performing "sum of products" calculations that are especially useful in certain types of digital signal processing for audio signals such as, for example, audio decompression, wavetable resampling, synthesis and filtering. Digital spatial and/or frequency filtering with a relatively large number of taps can be accommodated without loss of precision because of the 48-bit-wide accumulators contained with vector unit data paths 423. As one example of a particular optimum usage of vector unit 420 for audio processing, the eight separate register files 422 and associated data paths 423 of signal processor vector unit 420 can be used to simultaneously process eight different MIDI voices in parallel. The following are examples of additional audio processing that can be efficiently performed using vector unit 420:

- solving polynomial equations,
- processing 8 audio voices or 8 time samples in parallel,
- wavetable synthesis using cubic interpolation, wherein four of the vector unit data paths 423 are used to process one sample, and the other four vector unit data paths are used to process a second sample,
- audio enveloping processing wherein the 8 vector unit data paths can each multiply a different audio sample by a different weighting factor, and
- audio mixing processing wherein the 8 vector unit data paths can each multiply a different audio sample by a corresponding mixer weighting factor.

Because signal processor 400 can perform audio digital signal processing efficiently at high speed, it takes the signal processor only a small fraction of an audio

整理番号=000568

(133)

playback real time interval to perform and complete the digital audio processing associated with that time interval. For example, signal processor 400 takes much less than 1/30th of a second to digitally process audio that coprocessor audio interface 208 will playback in real time over a 1/30th of a second time interval. Because of this capability, signal processor 400 in this example can be time-shared between graphics processing and digital audio processing.

Generally, main processor 100 gives signal processor 400 a task list 250 at the beginning of a video frame that specifies the image and sound to be produced during the next succeeding video frame. Coprocessor 200 must be finished with both the audio and graphics processing for this next succeeding frame by the time that next succeeding frame begins. Because video display and audio playback is a real time continuous process (i.e., a new video image must be provided each video frame time, and audio must be continuously provided), coprocessor 200 needs to finish all audio and video signal processing associated with each next succeeding video frame by the time that next frame begins.

In this example, signal processor 400 is shared between graphics processing and digital audio signal processing. Because of the high speed calculating capabilities of signal processor vector unit 420, signal processor 400 is able to complete processing of the audio to be played during the next succeeding video frame in much less than the current video frame time, and is also able to complete graphics processing for the image to be displayed during the next succeeding image in less than the current frame time. This allows task list 250 to specify both graphics display lists and audio play lists that all must be completed by signal processor 400/coprocessor 200 by the beginning of the next video frame time. However, in this example there is nothing to prevent main processor

整理番号=000568

(134)

100 from giving coprocessor 200 a task list 250 that the coprocessor cannot complete before the next video frame begins. If the combined audio and graphics processing required by signal processor 400 is sufficiently intensive and time-consuming, the signal processor 400 can work on processing the task list for the entire current video frame time and still not be done by the beginning of the next video frame. It is up to video game program 108 to avoid overtaxing coprocessor 200, and to handle any overtaxing in an appropriate manner should it occur. A video game programmer can avoid overtaxing signal processor 400 by ensuring that all display lists 110 are organized efficiently, modeling the objects in 3-D in an efficient manner, and taking precautions to ensure that extensive time consuming processing (e.g., clipping) is avoided or minimized. Even with such precautions, however, it may take coprocessor 200 more than a single video frame time to complete especially complicated images. A video game programmer can handle this situation by slowing down the effective frame rate so that television 58 redisplay the same image stored in one part of frame buffer 118 for multiple video frames during which time coprocessor 200 can complete processing the next image. Because the user may perceive a variable frame rate as undesired delay, it is often best to slow down the overall effective frame rate to the rate required for coprocessor 200 to complete the most processing-intensive images—thus preventing more complex images from appearing more slowly than less complex images.

With respect to audio processing, it is generally unacceptable to fail to provide audio for a given video frame time since the user will hearing a disturbing "click" in a stream of otherwise continuous audio. Such audio disruptions are easily heard and can be annoying. Therefore, they should be avoided. One way to avoid an easily detectable audio disruption in a situation where signal processor 400 has failed to complete its

整理番号=000568

(135)

assigned audio processing in time is for main processor 100 to command audio interface 208 to replay the last frame's worth of audio during the next succeeding frame.

Acceptable audio can be produced in this way without the user noticing a disruption if done carefully. Other strategies include having signal processor 400 process multiple video frames worth of audio within a single video frame time—thereby providing an effective audio “frame” rate that is different (faster) than the effective video frame rate.

By “effective frame rate” we mean the rate at which coprocessor 200 produces a frame's worth of information (in this example, the television actual video frame rate stays constant).

[Example Audio Software Architecture]

Figure 24 shows an example of the overall software architecture provided by system 50 to synthesize and manipulate audio. This overall software architecture 700 includes four software objects, in this example a sequence player 702, a sound player 704, a synthesis driver 706 and audio synthesis microcode 708. In this example, sequence player 702, sound player 704, and synthesis driver 706 all execute on main processor 100, and audio synthesis microcode 708 runs on coprocessor signal processor 400. Thus, sequence player 702, sound player 704 and synthesis driver 706 are each supplied as part of game program 108 of storage device 54, and audio synthesis microcode 708 is supplied as part of SP microcode 156.

Sequence player 702, sound player 704 and synthesis driver 706 may differ depending on the particular video game being played. In general, sequence player 702 is responsible for the playback of Type 0 MIDI music sequence files. It handles sequence, instrument bank and synthesizer resource allocation, sequence interpretation, and MIDI

整理番号=000568

(136)

message scheduling. Sound player 704 is responsible for the playback of all ADPCM compressed audio samples. It is useful for sound effects and other streamed audio. Synthesis driver 706 is responsible for creating audio play lists 110 which are packaged into tasks by main processor 100 under software control and passed to coprocessor 200 in the form of task lists 250. In this example, synthesis driver 706 allows sound player 704 or other "clients" to assign wave tables to synthesizer voices, and to control playback parameters. As discussed above, the audio synthesis microcode 708 processes tasks passed to it and synthesizes L/R stereo 16-bit samples, which signal processor 400 deposits into audio buffers 114 within main memory 300 for playback via audio interface 208, audio DAC140 and amplifier/mixer 142.

In this example, synthesis driver 706 passes audio tasks to signal processor 400 in the form of audio "frames." A "frame" is a number of audio samples—usually something close to the number of samples required to fill a complete video frame time at the regular video frame rate (for example, 30 or 60 Hz). Although television set 58 receives and processes audio signals in a continuous stream unconstrained by any video frame rate parameter (e.g., the television can generate audio during horizontal and vertical video blanking and retrace), system 50 in this example organizes audio processing in terms of video frame rate because signal processor 400—which is shared between audio and graphics processing—must operate in accordance with the video frame rate because the graphics related tasks it performs are tied to the video frame rate.

[Example Play List Processing]

Figure 25 shows an example of a simple signal processor play list process. The Figure 25 process is specified by a play list 110 generated by main processor 100 under

整理番号=000568(137)

control of video game program 108, and specified as part of a task list 250. Thus, the Figure 25 SP play list process is an example of an output of synthesis driver 706 that is provided to signal processor 400 in the form of an audio play list 110.

Because of the limited size of instruction memory 402 in this example, audio synthesis microcode 708 is generally not continuously resident within signal processor 400. Instead, the initialization microcode main processor 100 arranges to be loaded into instruction memory 402 (see Figure 18, block 604), ensures that the appropriate audio microcode routine is loaded into the instruction memory for audio processing (also ensures that the appropriate graphics microcode routine is loaded into the instruction memory for graphics processing). The steps shown in Figure 25 assume that the audio synthesis microcode 708 is resident within the signal processor instruction memory 402, and that the signal processor 400 is reading an audio play list 110 specifying the steps shown.

Generally, the first task of an audio play list 110 is to set up buffers within signal processor data memory 408 required to perform the audio processing task (Figure 25, block 710). Generally, this buffer set up process involves allocating areas within data memory 404 to be used as one or more audio input buffers, and allocating an audio output buffer within the data memory. Generally, main processor 100 also commands signal processor 400 to use its DMA facility 454 to retrieve audio input data 112b from main memory into the allocated input buffer(s) for processing. Main processor 100 may next set certain attributes (e.g., volume ranges and change rates) to be used for the audio processing (Figure 25, block 712). Main processor 100 then specifies the types of signal processing to be performed by signal processor 400 along with appropriate parameters (Figure 25, block 714). In this example, main processor 100 can specify decompression,

整理番号=000568(138)

resampling, envelope/pan, mixing, and other processing (e.g., reverb) to be performed individually or in combination. The audio play list 110 typically will terminate with a command to save the contents of the output audio buffer stored in signal processor data memory 404 into main memory 300 (block 716).

[Example Audio Synthesis Microcode]

Figure 26 shows the overall tasks performed by audio synthesis microcode 708 in this example. Signal processor 400 under microcode control retrieves the next play list command from the current audio play list 110, and determines what kind of command it is (Figure 26, block 718). In this example, the audio command within an audio play list 110 may fall into the following general types:

- buffer command
- flow control command
- attribute command
- decompress command
- resample command
- envelope/pan command
- mix command
- special signal processing/effects command.

[Buffer Command Processing]

Buffer commands manage audio buffers within signal processor data memory 404, and permit audio data to be transferred between the data memory and main memory 300. The following are examples of buffer command formats and associated functions:

整理番号=000568

(139)

[Example Buffer Commands]**A_SETBUFF:**

	command		dmemin	
	dmemout		count	

This command sets the internal signal processor data memory 404 buffer pointers and count value used by the processing commands. This command is typically issued before any processing command. *dmemin* points to an input buffer, *dmemout* to an output buffer and *count* defines the number of 16 bit samples to process.

A_LOADBUFF:

	command		
	seg		address

This command loads a signal processor data memory 404 buffer from the main memory 300 address given by the *seg+address* fields. The SP data memory buffer location and the number of 16 bit samples to load are defined by issuing an A_SETBUFF command prior to the A_LOADBUFF command.

A_CLEARBUFF:

	command		dmemin	
			count	

This command clears an area of size *count* 16 bit samples starting at the signal processor 400 data memory address given by *dmem*.

整理番号=000568

(140)

A_SAVEBUFF:

	command			
	seg		address	

This command saves a buffer of 16 bit samples in the signal processor data memory 404 to the main memory 300 address given by the seg+address field. The input SP data memory buffer and number of samples are defined by issuing a A_SETBUFF command.

A_SEGMENT:

	command			
	seg		address	

See graphics G-SEGMENT command. This command is used to map indirect "segment" addresses into main memory 300 physical addresses.

Referring again to Figure 26, signal processor audio synthesis microcode 708 performs the specified buffer command by establishing, managing, writing data into, or reading data from the associated data memory buffer 409 (Figure 26, block 720).

Typically, signal processor 400 may use its DMA facility 454 to transfer data between main memory 300 and signal processor memory 404 in order to retrieve audio input data for processing or save audio data into main memory for playback by audio interface 208.

[Flow Control Command Processing]

If the next play list command is a flow control command, signal processor 400 responds to the command by traversing the current audio play list in the manner specified by the command. Nesting of audio play lists 110 is preferably permitted, and signal

整理番号= 0 0 0 5 6 8

(141)

processor 400 may maintain an audio play list stack in main memory 300 (just as it may do for graphics display lists).

[Attribute Command Processing]

If the next audio play list command is an attribute command, signal processor 400 processes the command by establishing appropriate mode and/or attribute conditions to be used for subsequent audio processing (Figure 26, block 724). In this example, audio synthesis microcode 708 supports the following example attribute command format and associated function:

[Example Attribute Commands]

A_SETVOL:

	command		volume	
	volume target		volume rate	

This command is used to set the volume parameters for subsequent processing commands. Currently this should be issued prior to A_ENVELOPE, A_PAN and A_RESAMPLE.

[Decompress Command Processing]

If the next audio play list command retrieved by signal processor 400 is a decompression command, the signal processor performs a decompression operation to decompress a compressed audio binary stream stored in an input buffer within data memory 404 to produce 16-bit audio samples which it stores in a defined audio output buffer within its data memory (Figure 26, block 726). In this example, audio synthesis

整理番号=000568

(142)

microcode 708 supports the following audio decompression command format and associated function:

[Example Decompression Command]

A_ADPCM:

	command	flags	gain	
	seg		address	

This command decompresses a binary stream in signal processor data memory 404 to produce 16 bit samples. The addresses in the data memory 404 for the input and output buffers and the number of samples to process are defined by issuing a A_SETBUFF command prior to the A_ADPCM command. The *seg+address* field points to a main memory 300 location which is used to save and restore state. The *gain* parameter is used to scale the output and is represented as S.15.

The *flags* define the behavior of the command. Currently defined *flags* are:

A_INIT, The *seg+address* field is used to restore state at the beginning of the command. If not set the pointer to state is ignored upon initiation, however, state is saved to this address at the end of processing.

A_MIX, The results are mixed into the output buffer. If not set results are put into the output buffer.

[Resample Command Processing]

整理番号=000568

(143)

If the next audio play list command signal processor 400 reads is a resample command, then the signal processor provides pitch shifting/resampling as well as integral envelope modulation based on the parameters specified in the command (Figure 26, block 728). The following is an example of a resample command and associated function supported by audio synthesis microcode 708.

[Example Resample Command]

A_RESAMPLE:

	command	flags	pitch	
	seg		address	

This command provides pitch shifting/resampling as well as integral envelope modulation. The signal processor data memory 404 input and output buffers and the number of samples are defined by issuing an A_SETBUFF command, and the volume envelope parameters are defined by issuing an A_SETVOL command. Resampling factor is defined by *pitch*.

The *flags* define the behavior of the command. Currently defined flags are:

A_INIT, The *seg+address* field is used to restore state at the beginning of the command. If not set the pointer to state is ignored upon initiation, however, state is saved to this address at the end of processing.

整理番号=000568

(144)

A_MIX, The results are mixed into the output buffer. If not set results are put into the output buffer.

[Envelope/Pan Command Processing]

If the next audio play list command signal processor 400 reads is an envelope/pan command, the signal processor performs that command by modulating one or two audio signal streams using a linear envelope (Figure 26, block 730). An envelope command multiplies an audio input sample stream by a linear function, and is thus able to ramp the volume of the audio up or down. A "pan" command generally applies inverse linear functions to audio in left and right stereo channels—accomplishing the effect of moving the perceived source of a sound or voice in space (i.e., from left to right or from right to left). The following examples of envelope/pan command formats and associated functions are supported by audio synthesis microcode 708 in this example of system 50.

[Example Envelope/Pan Commands]

A_ENVELOPE:

	command	flags		
	seg		address	

This command modulates a sample stream using a linear envelope. The parameters for the volume envelope are defined by issuing A_SETVOL and the signal processor data memory 404 buffer locations and number of samples to process are defined by issuing an A_SETBUFF prior to issuing the A_ENVELOPE command.

The *flags* define the behavior of the command. Currently defined flags are:

整理番号=0 0 0 5 6 8

(145)

- A_INIT,** The *seg+address* field field is used to restore state at the beginning of the command. If not set the pointer to state is ignored upon initiation, however, state is saved to this address at the end of processing.
- A_MIX,** The results are mixed into the output buffer. If not set results are put into the output buffer.

A_PAN:

	command	flags	dmemout2	
	seg		address	

This command provides 1 input, 2 output panning. Input, first output and number of samples are defined by issuing an A_SETBUFF command and the panning parameters are defined by issuing an A_SETVOL command. The second output is defined by *dmemout2*.

The *flags* defined the behavior of the command. Currently defined flags are:

- A_INIT,** The *seg + address* field field is used to restore state at the beginning of the command. If not set the pointer of state is ignored upon initiation, however, state is saved to this address at the end of processing.
- A_MIX,** The results are mixed into the output buffer. If not set results are put into the output buffer.

[Mixing Command Processing]

If the next audio play list command is a mixing command, signal processor 400 performs a mixing function to mix two audio input sample streams into the output audio buffer (Figure 26, block 732). The following example mixing command format and associated function is supported by signal processor 400 and audio synthesis microcode 708 in this example.

整理番号=000568

(146)

[Example Mixer Command]

A_MIXER:

	command		gain	
			dmemoutf	

This command provides a double precision mixing function. The single precision input is added to the double precision output after multiplication by *gain*. *dmemoutf* points to a signal processor data memory 404 area which stores the fractional part of the mixed stream. The input buffer, number of samples and integer part of the mixed output are defined by issuing an A_SETBUFF prior to the A_MIX.

[Special Audio Effects Processing]

If the next audio play list command is a special signal processing/effects command, signal processor 400 executes the command by providing the specified special effect or signal processing (Figure 26, block 734). An example special signal processing/effect is the addition of reverberation to create presence. This special effect simulates sound reflection in caves, concert halls, etc., and can also be used for various other special effects. Signal processor 400 and audio synthesis microcode 708 supports the following example reverberation special effects command format and associated function:

[Example Effects Command]

A_REVERB:

	command	flags		
	seg		address	

This command applies the reverb special effect to a sample stream. Signal processor data memory 404 input, output and number of samples are defined by issuing an A_SETBUFF command.

整理番号=000568

(147)

The *flags* define the behavior of the command. Currently defined flags are:

- A_INIT,** The *seg+address* field field is used to restore state at the beginning of the command. If not set the pointer to state is ignored upon initiation, however, state is saved to this address at the end of processing.
- A_MIX,** The results are mixed into the output buffer. If not set results are put into the output buffer.

[Audio Processing Structure]

To accomplish each of audio processing functions 728, 730, 732, 734 in this example, audio synthesis microcode 708 uses a general purpose effects implementation that manipulates data in a single delay line. Figure 27 shows an example general purpose audio processing implementation 740. In this example, the audio input samples can be conceived of as being applied to the input of contiguous single delay line 742. The output tap of the delay line is applied through a gain 744 to the audio output buffer within signal processor data memory 404. Samples from another tap on delay line 742 are passed through a summer 746 and returned to the delay line directly (over path 748) and also through a coefficient block 750, another summer 752 and a low pass filter 754. A further tap 756 from delay line 742 is connected to the other input of summer 752 and also to the other input of summer 746 (this time through a further coefficient block 758). This generalized implementation 740 allows a particular effect to be constructed by attaching an arbitrary number of effect primitives to single delay line 742. The parameters for each primitive in the effect are passed through via the commands discussed above. Each primitive consists of an all-pass with a variable length tap followed by a DC normalize (unity gain at DC) single pole low-pass filter 754 followed by an output gain 744 specifying how much of this primitive's output is to be contributed to the final effect output. The value of each of the parameters for a primitive specifies the

整理番号=000568

(148)

function of that primitive as a whole within the effect. Note that in Figure 27, the feedback coefficient 758 can be used to construct an "all-pass inside a comb" reverb (in response to the a_reverb command discussed above).

The general nature of implementation 740 does not mean that all functions are implemented. Only those functions which are driven by legitimate parameters actually generate audio command operations by signal processor 400. This gives video game programmers a great degree of flexibility in defining an effect that is appropriate in terms of both sonic quality and efficiency.

[COPROCESSOR DISPLAY PROCESSOR 500]

Display processor 500 in this example rasterizes triangles and rectangles and produces high quality pixels that are textured, anti-aliased and z-buffered. Figure 28 shows the overall processes performed by display processor 500. Display processor 500 receives graphics display commands that, for example, specify the vertices, color, texture, surface normal and other characteristics of graphics primitives to be rendered. In this example, display processor 500 can render lines, triangles, and rectangles. Typically, display processor 500 will receive the specifications for the primitives it is to render from signal processor 400, although it is also possible for main processor 100 to specify these commands directly to the display processor.

The first operation display processor 500 performs on an incoming primitive is to rasterize the primitive, i.e., to generate pixels that cover the interior of the primitive (Figure 28, block 550). Rasterize block 550 generates various attributes (e.g., screen location, depth, RGBA color information, texture coordinates and other parameters, and a coverage value) for each pixel within the primitive. Rasterize block 550 outputs the texture coordinates and parameters to a texture block 552. Texture block 552 accesses texture information stored within texture memory 502, and applies ("maps") a texel (texture element) of a specified texture within the texture memory onto each pixel outputted by rasterized block 550. A color convert block 554 and a chroma keying block

整理番号=000568(149)

556 further process the pixel value to provide a texture color to a color combined block 558.

Meanwhile, rasterize block 550 provides a primitive color (e.g., as a result of shading) for the same pixel to color combine block 558. Color combined block 558 combines these two colors to result in a single pixel color. This single pixel color output may have fog applied to it by block 560 (e.g., to create the effect of a smoke filled room, or the less extreme, natural effect of reducing color brilliance as an object moves further away from the viewer). The resulting pixel color value is then blended by a block 562 with a pixel value framebuffer 118 stores for the same screen coordinate location. An additional anti-alias/z-buffer operation 564 performs hidden surface removal (i.e., so closer opaque objects obscure objects further away), anti-aliasing (to remove jaggedness of primitive edges being approximated by a series of pixels), and cause the new pixel value to be written back into framebuffer 118.

The operations shown in Figure 28 are performed for each pixel within each primitive to be rendered. Many primitives may define a single complex scene, and each primitive may contain hundreds or thousands of pixels. Thus, display processor 500 must process millions of pixels for each image to be displayed on color television set 58.

Typically, framebuffer 118 is "double buffered" -- meaning that it is sized to contain two complete television screen images. Display processor 500 fills one screen worth of framebuffer information while video interface 210 reads from the other half of the framebuffer 118. At the end of the video frame, the video interface 210 and display processor 500 trade places, with the video interface reading from the new image representation just completed by display processor 500 and the display processor rewriting the other half of the framebuffer. This double buffering does not give display processor 500 any more time to complete an image; it must still finish the image in nominally one video frame time (i.e., during the video frame time just prior to the frame time during which the new image is to be displayed).

[Pipelining]

整理番号=000568

(150)

Because high speed operation is very important in rendering pixels, display processor 500 has been designed to operate as a "pipeline." Referring again to Figure 28 "pipelining" means that the various steps shown in Figure 28 can be performed in parallel for different pixels. For example, rasterize block 550 can provide a first pixel value to texture block 552, and then begin working on a next pixel value while the texture block is still working on the first pixel value. Similarly, rasterize block 550 may be many pixels ahead of the pixel that blend block 562 is working on.

In this example, display processor 500 has two different pipeline modes: one-cycle mode, and two-cycle mode. In one-cycle mode, one pixel is processed for each cycle time period of display processor 500. A one-cycle mode operation is shown in Figure 29(a). Note that the operations shown in Figure 29(a) are themselves pipelined (i.e., the blend operation 562 operates on a different pixel than the rasterize operation 550 is currently rasterizing), but the overall operation sequence processes one pixel per cycle.

Figure 29(b) shows the two-cycle pipeline mode operation of display processor 500 in this example. In the Figure 29(b) example, some of the operations shown in Figure 28 are performed twice for each pixel. For example, the texture and color convert/filtering operations 552, 554 shown in Figure 28 are repeated for each pixel; the color combine operation 558 is performed twice (once for the texture color output of one texture operation, and once for the texture color output of the other texture operation). Similarly, blend operation 562 shown in Figure 28 is performed twice for each pixel.

Even though these various operations are performed twice, display processor 500 in this example does not contain duplicate hardware to perform the duplicated operations concurrently (duplicating such hardware would have increased cost and complexity). Therefore, in this example, signal processor 500 duplicates an operation on a pixel by processing it with a particular circuit (e.g., a texture unit, a color combiner or a blender), and then using the same circuit again to perform the same type of operation again for the same pixel. This repetition slows down the pipeline by a factor of two (each pixel must "remain" at each stop in the pipeline for two cycles instead of one), but allows more complicated processing. For example, because the two-cycle-per-pixel mode can map

整理番号=000568

(151)

two textures onto the same pixel, it is possible to do "trilinear" ("mipmapping") texture mapping. In addition, since in this example, display processor 500 uses the same blender hardware to perform both the fog operation 560 and the blend operation 562 (but cannot both blend and fog simultaneously), it is generally necessary to operate in the two-cycle-per-pixel mode to provide useful fog effects.

The following tables summarize the operations performed by the various blocks shown in Figures 29(a) and 29(b) during the one-cycle and two-cycle modes:

Display Processor Pipeline Block Functionality in One-Cycle Mode	
Block	Functionality
Rasterize 550	Generates pixel and its attribute covered by the interior of the primitive.
Texture 552	Generates 4 texels nearest to this pixel in a texture map.
Filter Texture 554	Bilinear filters 4 texels into 1 texel, OR performs step 1 of YUV-to-RGB conversion.
Combine 558	Combines various colors into a single color, OR performs step2 of YUV-to-RGB conversion.
Blend 562	Blends the pixel with framebuffer memory pixel, OR fogs the pixel for writing to framebuffer.
Framebuffer 563	Fetches and writes pixels (color and z) from and to the framebuffer memory.

整理番号=000568

(152)

Display Processor Pipeline Block Functionality In Two-Cycle Mode	
Block	Functionality
Rasterize 550	Generates pixel and its attribute covered by the interior of the primitive.
Texture 552a	Generates 4 texels nearest to this pixel in a texture map. This can be level X of a mipmap.
Texture 552b	Generates 4 texels nearest to this pixel in a texture map. This can be level X+1 of a mipmap.
Filter Texture 554a	Bilinear; filters 4 texels into 1 texel.
Filter Texture 554b	Bilinear; filters 4 texels into 1 texel.
Combine 558a	Combines various colors into a single color, OR linearly interpolates the 2 bilinear filtered texels from 2 adjacent levels of a mipmap, OR performs step 2 of YUV-to-RGB conversion.
Combine 558b	Combines various colors into a single color, OR chroma keying.
Blend 562a	Combines fog color with resultant CC1 color.
Blend 562b	Blends the pipeline pixels with framebuffer memory pixels.
Framebuffer 563a	Read/modify/write color memory; and
Framebuffer 563b	Read/modify/write Z memory.

[Fill and Copy Operations]

Display processor 500 also has a "fill" mode and a "copy" mode, each of which process four pixels per cycle. The fill mode is used to fill an area of framebuffer 118 with

整理番号= 0 0 0 5 6 8

(153)

identical pixel values (e.g., for high performance clearing of the framebuffer or an area of it). The copy mode is used for high-performance image-to-image copying (e.g., from display processor texture memory 502 into a specified area of framebuffer 118). The copy mode provides a bit "blit" operation in addition to providing high performance copying in the other direction (i.e., from the framebuffer into the texture memory).

The pipeline operations shown in Figures 29(a) and 29(b) are largely unused during the fill and copy modes, because in this example, the operations cannot keep up with the pixel fill or copy rate. However, in this example, an "alpha compare" operation (part of blend operation 562) is active in the copy mode to allow display processor 500 to "blit" an image into framebuffer 118 and conditionally remove image pixels with the word alpha=0 (e.g., transparent pixels).

The display processor's mode of operation is selected by sending the display processor 500 a "set other mode" command specifying a "cycle type" parameter. See Figure 86. In the one-cycle-per-pixel or two-cycle-per-pixel pipeline modes, additional display processor 500 commands are available to insure that pipeline synchronization is maintained (e.g., so that the pipeline is emptied of one primitive before the parameters of another primitive take effect). See Figure 100.

[EXAMPLE DISPLAY PROCESSOR 500 ARCHITECTURE]

Figure 30 shows an example architecture of display processor 500. In this example, display processor 500 includes a command unit 514 with associated RAM 516 and DMA controller 518; an "edge walker"/rasterizer 504; a RGBAZ pixel stepper 520; a color combiner/level interpreter 508, a blender/fogger 510, a ditherer 522, a coverage evaluator 524, a depth (z) comparator 526, a memory interface 512 and a texture unit 506. In this case, texture unit 506 includes, in addition to texture memory 502, texture steppers 528, a texture coordinate unit 530 and a texture filter unit 532.

Command unit 514 and DMA controller 518 connect to coprocessor main internal bus 214, and also connect to the signal processor 400 via a private "x" bus 218. Memory interface 512 is a special memory interface for use by display processor 500 primarily to

整理番号=000568(154)

access to the color framebuffer 118a and the z buffer 118b stored within main memory 300 (thus, display processor 500 has access to main memory 300 via memory interface 512 and also via coprocessor internal bus 214).

[DMA Controller]

DMA controller 518 receives DMA commands from signal processor 400 or main processor 100 over bus 214. DMA controller 518 has a number of read/write registers shown in Figures 31(a)-31(c) that allow signal processor 400 and/or main processor 100 to specify a start and end address in SP data memory 404 or main memory 300 from which to read a string of graphics display commands (Figure 31(a) shows a start address register 518A, and Figure 31(b) shows an end address register 518B). DMA controller 518 reads data over main coprocessor bus 214 if registers 518a, 518b specify a main memory 300 address, and it reads data from the signal processor's data memory 404 over private "x bus" 214 if the registers 518a, 518b specify a data memory 404 address. DMA controller 518 also includes a further register (register 518C shown in Figure 31(c)) that contains the current address DMA controller 518 is reading from. In this example, DMA controller 518 is uni-directional -- that is, it can only write from bus 214 into RAM 516. Thus, DMA controller 518 is used in this example for reading from signal processor 400 or main memory 300. In this example, display processor 500 obtains data for its texture memory 502 by passing texture load commands to command unit 514 and using memory interface 512 to perform those commands.

[Command Unit]

Command unit 514 retains much of the current state information pertaining to display processor 500 (e.g., mode and other selections specified by "set commands"), and outputs attributes and command control signals to specify and determine the operation of the rest of display processor 500. Command unit 514 includes some additional registers that may be accessed by main processor 100 (or signal processor 400) via coprocessor bus 214. These additional registers, which are mapped into the

整理番号=000568

(155)

address space of main processor 100, permit the main processor to control and monitor display processor 500.

For example, command unit 514 includes a status/command register 534 shown in Figure 32(d) that acts as a status register when read by main processor 100 and acts as a command register when the main processor writes to it. When reading this register 534, main processor 100 can determine whether display processor 500 is occupied performing a DMA operation reading from signal processor data memory 404 (field 536(1); whether the display processor is stalled waiting for access to main memory 300 (field 536(2); whether the display processor pipeline is being flushed (field 536(3); whether the display processor graphics clock is started (field 536(4); whether texture memory 502 is busy (field 536(5); whether the display processor pipeline is busy (field 536(6); whether command unit 514 is busy (field 536(7); whether the command buffer RAM 516 is ready to accept new inputs (field 536(8); whether DMA controller 518 is busy (field 536(9); and whether the start and end addresses and registers 518a and 518b respectively valid (fields 536 (10), 536(11). When writing to this same register 534, main processor 100 (or signal processor 400) can clear an X-bus DMA operation from the signal processor 400 (field 538 (1); begin an X-bus DMA operation from signal processor data memory 404 (field 538(2); start or stop the display process (fields 538 (3), 538(4); start or stop a pipeline flushing operation (fields 538 (5), 538(6); clear a texture memory address counter 540 shown in Figure 33(h) (field 538 (7); clear a pipeline busy counter 542 shown in Figure 33(f) (field 538(8); clear a command counter 544 used to index command buffer RAM 516 (field 538(9) (the counter 544 is shown in Figure 33(g); and clear a clock counter 546 (see Figure 33(e)) used to count clock cycles (field 538 (10).

As mentioned above, the clock count, buffer count, pipeline count and texture memory count can all be read directly from registers 540-546 (see Figures 33(e)-33(h)). In addition, main processor 100 or signal processor 400 can read and control the BIST operation pertaining to texture memory 502 (see BIST status/control register 548 shown in Figure 34(i)), and can also enable in control testing of memory interface 512 by manipulating mem span test registers 549(a), 549(b) and 549(c) shown in Figure 34(j).

整理番号=000568

(156)

Referring back to Figure 30, once one or more commands have been loaded into command unit buffer ram 518 and display processor 500 has been started, command unit 514 begins reading and processing each command sequentially. The repertoire of commands display processor 500 understands are shown Figures 65-102. Hardware (e.g., logic, gate arrays and the like) within display processor 500 directly interpret the graphics display commands within RAM 516. In this example, display processor 500 has no ability to branch or jump in traversing this list of commands. Rather, display processor 500 in this example is a sequential state machine that accepts each new command as an input in strict sequence and alters its states and outputs in response to the command.

Display processor 500 halts if its command buffer RAM 516 is empty (i.e., it has processed all of the commands in the buffer, which buffer acts as a FIFO). Main processor 100 or signal processor 400 can determine if display processor 500 has halted by reading display processor status register 534 and may, if desired, pass the display processor a command that stalls the display processor temporarily (see "Sync Full" command in Figure 98).

[Edgewalker and Steppers]

Edgewalker 504 shown in Figure 30 performs the rasterize process 550 shown in Figure 28. In this example, edgewalker 504 receives the edge coefficients, shade coefficients, texture coefficients and z buffer coefficients specified in a "triangle command" (see Figure 73 specifying a particular primitive open line, triangle or rectangle), and outputs "span" values from which the following attributes for each pixel enclosed within the primitive can be derived:

- screen x, y location
- z depth for z buffer purposes
- RGBA color information
- s/w, t, w, 1/w texture coordinates, level-of-detail for texture index, perspective correction, and mipmapping (these are commonly referred to s, t, w, 1)

整理番号=0 0 0 5 6 8

(157)

- coverage value (pixels on the edge of a primitive have partial coverage values, whereas pixels within the interior of a primitive are full).

Edgewalker 504 sends the parameters for a line of pixels across the primitive (a "span") to the pipeline hardware downstream for other computations. In particular, texture steppers 528 and RGBAZ steppers 520 receive the "span" information specified by edgewalker 504, and step sequentially along each pixel in the horizontal line (in the view plane coordinate system) of the "span" to derive the individual texture coordinates and RGBAZ values for each individual pixel in the span.

The RGBAZ stepper 520 may also perform a "scissoring" operation on triangle primitives (this does not work for rectangles in this example) to efficiently eliminate portions of triangle primitives extending outside of a view plane scissoring rectangle. Scissoring is commonly used to eliminate running performance-intensive clipping operations on signal processor 400. Scissoring is similar in concept to clipping, but whereas clipping is performed in the 3-D coordinate system, scissoring is performed in the 2-D coordinate system of the viewing plane. Scissoring by steppers 520, 528 is invoked by sending display processor 500 a "set scissor" command (see Figure 94).

As mentioned above, steppers 520 produces color and alpha information for each pixel within the "span" defined by edgewalker 504. Similarly, texture steppers 528 produces texture coordinate values (s, t, w) for each pixel within the span. Steppers 520, 528 operate in a synchronized fashion so that texture unit 506 outputs a mapped texture value for a pixel to color combiner 58 at the same time that the RGBAZ steppers 520 output a color value for the same pixel based on primitive color, shading, lighting, etc.

[Texture Unit]

Texture unit 506 in this example takes the texture coordinates s, t, w and level-of-detail values for a pixel (as mentioned above, texture steppers 528 derive these values for each individual pixel based upon "span" information provided by edgewalker 504), and fetches appropriate texture information from onboard texture memory 502 for mapping onto the pixel. In this example, the four nearest texels to the screen pixel are fetched

整理番号=000568

(158)

from texture memory 502, and these four texel values are used for mapping purposes. Video game program 108 can manipulate texture states such as texture image types and formats, how and where to load texture images, and texture sampling attributes.

Texture coordinate unit 530 computes appropriate texture coordinates for mapping texture stored within texture memory 502 onto the primitive being rendered. Since the 2-dimensional textures stored in texture memory 502 are square or rectangular images that must be mapped onto triangles of various sizes, the texture coordinate in 530 must select appropriate texels within the texture to map onto pixels in the primitive to avoid distorting the texture. See OpenGL Programming Guide at 278.

Texture coordinate unit 530 computes a mapping between the inputted pixel texture coordinates and four texels within the appropriate texture stored in texture memory 502. Texture coordinate unit 530 then addresses the texture memory 502 appropriately to retrieve these four texels. The four texel values are passed to the texture filter unit 532. Texture filter 532 takes the four texels retrieved from texture memory 502 and produces a simple bilinear-filtered texel. Texture filter 532 in this example can perform three types of filter operations: point sampling, box filtering, and bilinear interpolation. Point sampling selects the nearest texel to the screen pixel. In the special case where the screen pixel is always the center of four texels, the box filter can be used. In the case of the typical 3-D, arbitrarily rotated polygon, bilinear filtering is generally the best choice available. For hardware cost reduction, display processor texture filter unit 532 does not implement a true bilinear filter. Instead, it linearly interpolates the three nearest texels to produce the result pixels. This has a natural triangulation bias which is not noticeable in normal texture images but may be noticed in regular pattern images. This artifact can be eliminated by prefiltering the texture image with a wider filter. The type of filtering performed by texture filter unit 532 is set using parameters in the "set modes" display command (see Figure 65-102).

整理番号=000568

(159)

[TEXTURE MEMORY 502]

Display processor 500 treats texture memory 502 as a general-purpose texture memory. In this example, texture memory 502 is divided into four simultaneously accessible banks, giving output of four texels per clock cycle. Video game program 58 can load varying-sized textures with different formats anywhere in the texture memory 502. Texture coordinate unit 530 maintains eight texture tile descriptors that describe the location of texture images within texture memory 502, the format of each texture, and its sampling parameters. This allows display processor 500 to access as many as eight different texture tiles at a time (more than eight texture tiles can be loaded into the texture memory, but only eight tiles are accessible at any time).

Figure 35 shows an example of the texture tile descriptors and their relationship to texture tiles stored in texture memory 502. In this particular example shown in Figure 35, eight different texture tiles 802 are stored within texture memory 502. Each texture tile 802 has an associated texture tile descriptor block 804 (as discussed above, display processor 500 maintains up to eight descriptors 804 corresponding to eight texture tiles stored within texture memory 502). The texture descriptors contain information specified by a "set tile" command (see Figure 68). For example, these texture tile descriptors specify the image data format (RGBA, YUV, color index mode, etc.), the size of each pixel/texel color element (four, eight, sixteen, thirty-two bits), the size of the tile line in 64-bit words, the starting address of the tile in texture memory 502, a palette number for 4-bit color indexed texels, clamp and mirror enables for each of the S and T directions, masks for wrapping/mirroring in each of S and T directions, level of detail shifts for each of S and T addresses. These descriptors 804 are used by texture coordinate unit 530 to calculate addresses of texels within the texture memory 502.

[Texture Coordinate Unit]

Figure 36 shows a more detailed example of the processing performed by texture coordinate unit 530. Figure 36 shows the various tile descriptors 804 being applied as inputs to texture coordinate unit 530. Figure 36 also shows that texture coordinate unit

整理番号=000568

(160)

530 receives the primitive tile/level/texture coordinates for the current pixel from texture steppers 528. Texture coordinate unit 530 additionally receives mode control signals from command unit 514 based, for example, on the "set other mode" and "set texture image" commands (see Figure 86, Figure 87 and Figure 66). Based on all of this input information, texture coordinate unit 530 calculates which tile descriptor 804 to use for this primitive, and converts the inputted texture image coordinates to tile-relative coordinates which the texture coordinate unit wraps, mirrors and/or clamps as specified by the tile descriptor 804. Texture coordinate unit 530 then generates an offset into texture memory 502 based on these tile coordinates. The texture coordinate unit 530 in this example can address 2×2 regions of texels in one or two cycle mode, or 4×1 regions in copy mode. Texture coordinate unit 530 also generates S/T/L fraction values that are used to bi-linearly or tri-linearly interpolate the texels.

Figure 37 is a detailed diagram of texture coordinate unit 530 and texture memory unit 502. As shown in Figure 37, the incoming s, t, w texture coordinates are inputted into a perspective correction block 566 which provides a perspective correction based on w when perspective correction is enabled. The perspective-corrected s, t values are then provided to a level-of-detail or precision shift block 568 which shifts the texture coordinates after perspective divide (e.g., for MIP mapping and possibly for precision reasons). A block 570 then converts the shifted texture coordinates to tile coordinates, providing fractional values to the texture filter unit 532. These tile coordinate values are then clamped, wrapped and/or mirrored by block 572 based on the current texture mode parameters of display processor 500. Meanwhile, the perspective-corrected texture coordinates provided by perspective correction block 566 are also provided to a level of detail block 574 which, when level of detail calculations are enabled, calculates a tile descriptor index into a tile descriptor memory 576 and also calculates a level of detail fractional value for interpolation by the color combiner 508. The tile descriptors 804 are stored in tile descriptor memory 576, and are retrieved and outputted to a memory conversion block 578 which conversion block also receives the adjusted texture coordinate values of block 572. Address conversion block 578 converts the adjusted

整理番号=000568

(161)

texture coordinate values into texture memory unit addresses based on current tile size, format and other parameters as specified by the tile descriptor 804. Address conversion block 578 outputs the texel address to texture memory unit 502. The texture memory unit 502 also receives additional parameters which are used, for example, if the texture is color indexed. Texture memory unit 502 outputs four texel values to texture filter unit 532 for filtering as discussed above.

[Texture Memory Loading]

Texture memory unit 502 includes a four kilobyte random access memory onboard coprocessor 200. Because texturing requires a large amount of random accesses with consistent access time, it is impractical to texture directly from main memory 300 in this example. The approach taken is to cache up to four kilobytes of an image in on-chip, high-speed texture memory 502. All primitives can be textured using the contents of texture memory 502.

In order to use texture memory 502, video game program 108 must load a texture tile into the texture memory and then load the associated descriptor 804 into tile descriptor 576. The "load tile" command (see Figure 69) is used to load a tile into texture memory 502, and a "set tile" and "set tile size" command are used to load corresponding tile descriptor blocks 804 into tile descriptor memory 576. In addition, a "Load Tlut" command (see Figure 72) can be used to load a color lookup table into texture memory 502 for use by color indexed textures.

Physically, texture memory 502 is organized in four banks, each comprising 256 16-bit wide words, each bank having a low half and a high half. This organization can be used to store 4-bit textures (twenty texels per row), 8-bit textures (ten texels per row), 16-bit textures (six texels per row), 16-bit YUV textures (twelve texels per row), and 32-bit textures (six texels per row). In addition, texture unit 506 in this example supports a color-indexed texture mode in which the high half of texture memory 502 is used to store a color lookup table and the low half of the texture memory is used to store 4-bit or 8-bit color indexed textures. This organization is shown in Figure 38. In this Figure 38

整理番号=000568

(162)

example, a color indexed texture tile 580 is stored in a low half 502(L) of texture memory 502, and a corresponding color lookup table 582 is stored in the upper half 502(H) of the texture memory.

Figure 39 shows a more detailed depiction of a particular texture memory color indexed mode, in which the color lookup table 582 is divided into four palette banks 584 or tables, each having, for example, sixteen entries, each entry being 16-bits wide. The color lookup table may represent color in 16-bit RGBA format, or in 16-bit IA format. Since four texels are addressed simultaneously, there are four (usually identical) lookup tables 484 stored in the upper half of texture memory 502. As mentioned above, these lookup tables are loaded using the "load Tlut" command shown in Figure 72.

Display processor 500 supports another color-indexed texture mode in which each texel in the lower half of texture memory 502 comprises eight bits—and therefore can directly access any one of the 256 locations in the upper half 502(H) of texture memory 502. Thus, 8-bit color-indexed textures do not use the palette number of the tile, since they address the whole 256-element lookup table directly. It is not necessary to use the entire upper half of texture memory 502 for a lookup table when using 8-bit color-indexed textures. For example, if less than eight of the bits of the 8-bit color-indexed texture tile is being used for color lookup, only a portion of color memory upper half 502(H) is required to store the lookup table—and the remainder of the upper half of the texture memory 502 might thus be used for storing a non-color-indexed texture such as a 4-bit I texture (see Figure 38). Similarly, even when color-indexed texture 580 is stored in the lower half 502(L) of texture memory 502, it is possible to also store non-color-indexed textures in the lower half as well. Thus, color-indexed textures and non-color-indexed textures can be co-resident in texture memory 502.

The following texture formats and sizes are supported by texture memory 502 and texture coordinate unit 530:

Texture Format and Sizes				
Type	4-bit	8-bit	16-bit	32-bit
RGBA			X	X
YUV			X	
Color Index	X	X		
Intensity Alpha (IA)	X	X	X	
Intensity (I)	X	X		

In this example, texture unit 506 will, unless explicitly told otherwise, change a tile descriptor 804 or a texture tile 802 immediately upon loading—even if it is still being used for texture mapping of a previous primitive. Texture loads after primitive rendering should be preceded by a “sync load” command and tile descriptor attribute changes should be preceded by a “sync tile” command to ensure that the texture tile and tile descriptor state of texture unit 506 does not change before the last primitive is completely finished processing (see Figure 99 and Figure 101 for example formats and functions of these commands).

As mentioned above in connection with the signal processor 400, two special commands (“texture rectangle” and “texture rectangle flip”) can be used to map a texture onto a rectangle primitive (see Figure 83 and Figure 84). It is possible to use the “texture rectangle” command to copy an image from texture memory 502 into frame buffer 118, for example. See Figure 83.

[COLOR COMBINER]

Referring once again to Figure 30, color combiner 508 combines texels outputted by texture unit 506 with stepped RGBA pixel values outputted by RGBAZ steppers 520. Color combiner 508 can take two color values from many sources and linearly interpolate between them. The color combiner 508 performs the equation:

$$newcolor = (A-B) * C + D$$

Here, A, B, C and D can come from many different sources (note that if $D = B$, then color combiner 508 performs simple linear interpolation).

Figure 40 shows possible input selection of a general purpose linear interpolator color combiner 508 for RGB and Alpha color combination in this example. As can be seen in Figure 40, only some of the inputs in the lefthand column come from texture unit 506 or RGBAZ steppers 520. The rest of the inputs are derived from color combiner 508 internal state that can be programmed by sending commands to display processor 500. As discussed above, the "combined color" and "combined Alpha" values provided to color combiner 508 are obtained from the RGBAZ steppers 520, and the texel color and texture Alpha are obtained from texture unit 506 (two texel colors and corresponding Alpha values are shown since in two-cycle-per-pixel mode two texels will be provided by texture unit 506 for purposes of mipmapping for example). Additionally, the level of detail fractional input is obtained from Figure 37 block 574, and the primitive level of detail value along with the primitive color and primitive Alpha value may be obtained from a "set primitive color" command sent to display processor 500 (see Figure 89) (the primitive color value/alpha/level of detail fraction value can be used to set a constant polygon face color). Similarly, a shade color and associated Alpha value may be obtained from a "shade coefficient" command (see Figure 77 and Figure 78), and an environment color and associated Alpha value may be obtained from a "set environment color" command (see Figure 88) (the environment color/alpha value described above can be used to represent the ambient color of the environment). Two kinds of "set key" commands (one for green/blue, the other for red) are used for green/blue color keying and red color keying respectively—these supplying the appropriate key:center and key:scale inputs to color combiner 508 (see Figure 96 and Figure 97). Both the primitive and environment values are programmable and thus can be used as general linear interpolation sources.

Convert K4 and K5 Inputs to color combiner 508 are specified in this example by the "set convert" command (see Figure 95) that adjust red color coordinates after conversion of texel values from YUV to RGB format (the remainder of the conversion

整理番号=000568

(165)

process responsive to this set convert command being performed within texture filter unit 532).

Figure 41 shows a portion of color combiner 508 used for combining the alpha values shown as inputs in Figure 40. For both the RGB color combine in alpha color combine operations performed by color combiner 508, there are two modes, one for each of the two possible pipeline modes (one cycle-per-pixel, and two cycles-per-pixel). In the two-cycle mode, color combiner 508 can perform two linear interpolation arithmetic computations. Typically, the second cycle is used to perform texture and shading color modulation (i.e., the operations color combiner 508 are typically used for exclusively in the one-cycle mode), and the first cycle can be used for another linear interpolation calculation (e.g., level of detail interpolation between two bi-linear filtered texels from two mipmap tiles). Color combiner 508 also performs the "alpha fix-up" operation shown in Figure 42 in this example (see "set key GB" command in Figure 97).

[Blender]

As discussed above, blender 510 takes the combined pixel value provided by color combiner 508 and blends them against the frame buffer 118 pixels. Transparency is accomplished by blending against the frame buffer color pixels. Polygon edge antialias is performed, in part, by blender 510 using conditional color blending based on depth (z) range. The blender 510 can also perform fog operations in two-cycle mode.

Blender 510 can perform different conditional color-blending and z buffer updating, and therefore can handle all of the various types of surfaces shown in Figure 43 (i.e., opaque surfaces, decal surfaces, transparent surfaces, and inter-penetrating surfaces).

An important feature of blender 510 is its participation in the antialias process. Blender 510 conditionally blends or writes pixels into frame buffer 118A based on depth range (see Figure 46 which shows example z buffer formats including a "dz" depth-range field). See copending Japanese Patent Application Reference No. 0569 filed by Nintendo Co., Ltd. and Silicon Graphics, Inc., entitled "System and Method For Merging Pixel Fragments Based On Depth Range Values".

整理番号=000568

(166)

In this example, video interface 210 applies a spatial filter at frame buffer read-out time to account for surrounding background colors to produce antialias silhouette edges. The antialiasing scheme requires ordered rendering sorted by surface or line types. Here is the rendering order and surface/line types for z buffer antialiasing mode:

1. All opaque surfaces are rendered.
2. All opaque decal surfaces are rendered.
3. All opaque interpenetrating surfaces are rendered.
4. All of the translucent surface and lines are rendered last.

These can be rendered in any order, but proper depth order gives proper transparency.

The mode blender 510 is controlled, in part by the groups of coefficients specified in the triangle command defining the primitive (see Figure 73). Thus, a primitive can be rendered in a z buffered mode or non-z buffered mode as specified by the triangle command. In addition, the "set other modes" command (see Figure 86 and Figure 87) specifies blend mode words for cycle 0 and cycle 1 in addition to specifying "blend masks" and enabling/disabling antialiasing.

Blender 510 has two internal color registers: fog color and blend color. These values are programmable using the "set fog color" and "set blend color" commands, respectively (see Figure 90 and Figure 91). These values can be used for geometry with constant fog or transparency.

Blender 510 can compare the incoming pixel alpha value with a programmable alpha source to conditionally update frame buffer 118A. This feature can allow complex, outlined, billboard type objects, for example. Besides thresholding against a value, blender 510 in this example can also compare against a dithered value to give a randomized particle effect. See "set other modes" command (Figure 86 and Figure 87). Blender 510 can also perform fog operations, either in 1-cycle or 2-cycle mode. Blender 510 uses the stepped z value as a fog coefficient for fog and pipeline color blending.

Figure 44 shows an example of the overall operations performed by blender 510 in this example. In this particular example, blender 510 can be operated in a mode in which a coverage value produced by coverage evaluator 524 can be used to specify the amount

整理番号=000568

(167)

of blending. Coverage evaluator 524 compares the coverage value of the current pixel (provided by edge walker 504) to stored coverage value within frame buffer 118A. As shown in Figure 45 (a depiction of the format of the color information stored for each pixel within color frame buffer 118A), the color of a pixel is represented by 5-bits each of red, green, and blue data and by a 3-bit "coverage" value. This "coverage" value can be used as-is, or multiplied by an alpha value for use as pixel alpha and/or coverage (see "set other modes" command in Figure 86 and Figure 87). The "coverage" value nominally specifies how much of a pixel is covered by a particular surface. Thus, the coverage value outputted by edge walker 504 will be 1 for pixels lying entirely within the interior of a primitive, and some value less than 1 for pixels on the edge of the primitive. In this example, blender 510 uses the coverage value for antialiasing. At the time blender 510 blends a primitive edge, it does not know whether the primitive edge is internal to an object formed from multiple primitives or whether the edge is at the outer edge of a represented object. To solve this problem in this example, final blending of opaque edge values is postponed until display time, when the video interface 210 reads out frame buffer 118A for display purposes. Video interface 210 uses this coverage value to interpolate between the pixel color and the colors of neighboring pixels in the frame buffer 118A. In order to accomplish this antialiasing at display time, blender 510 must maintain the coverage value for each pixel within frame buffer 118a, thereby allowing video interface 210 to later determine whether a particular pixel is a silhouette edge or an internal edge of a multi-polygon object.

[Memory Interface 512 and Z Buffering]

Memory interface 512 provides an interface between display processor 500 and main memory 300. Memory interface 512 is primarily used during normal display processor 500 operations to access the color frame buffer 118a and the Z buffer 118b. Color frame buffer 118a stores a color value for each pixel on color television screen 60. The pixel format is shown in Figure 45. Z buffer 118b stores a depth value and a depth range value for each color pixel value stored in color frame buffer 118a. An example

整理番号=000568

(168)

format for z buffer values is shown in Figure 46. The Z buffer 118b is used primarily by blender 510 to determine whether a newly rendered primitive is in front of or behind a previously rendered primitive (thereby providing hidden surface removal). The "DZ" depth range value shown in Figure 46 may be used to help ascertain whether adjacent texels are part of the same object surface.

Memory interface 512 can write to main memory 300, read from main memory, or read, modify and write (RMW) locations in the main memory. For RMW operations, memory interface 512, in this example, pre-fetches a row of pixels from frame buffer 118a as soon as edge walker 504 determines the x, y coordinates of the span. Memory interface 512 includes an internal "span buffer" 512a used to store this span or row of pixels. Memory interface 512 provides the appropriate pre-fetched pixel value from span buffer 510a to blender 510 at the appropriate time--thus minimizing the number of accesses to main memory 300. Span buffer 512a is also used to temporarily store blended (modified) pixel values so that display processor 500 need not access main memory 300 each time a new pixel value is blended. In general, memory interface 512 writes the entire span worth of pixels into main memory 300 as a block all at once.

Memory interface 512 has enough on-chip RAM to hold several span buffers. This can cause problems, however, if two spans in sequence happen to overlap the same screen area. A parameter "atomic space" in the "Set Other Modes" command (see Figure 86 and Figure 87) forces memory interface 512 to write one primitive to frame buffer 118a before starting the next primitive--thereby avoiding this potential problem by adding no cycles after the last span of a primitive is rendered.

Depth comparator 526 operates in conjunction with z buffer 118b to remove hidden surfaces and to insure the transparent values are blended properly. Depth comparator 526 compares the z or depth value of the current pixel with the z value currently residing in z buffer 118a for that screen location. At the beginning of the rendering of a new frame, all locations in z buffer 118b are preferably initialized to maximum distance from the viewer (thus, any object will be open "in front of" this initialized value). Generally, each time display processor 500 is to blend a new pixel into frame buffer 118a, depth

整理番号=000568

(169)

comparator 526 compares the depth of the current pixel with the depth residing in that location of z buffer 118b. If the old z buffer value indicates that the previously written pixel is closer to the viewer than is the new pixel, the new pixel is discarded (at least for opaque values) and is not written into the frame buffer--thus accomplishing hidden surface removal. If the new pixel is closer to the old pixel as indicated by depth comparator 526, then the new pixel value (at least for opaque pixels) may replace the old pixel value in frame buffer 118a--and the corresponding value in z buffer 118b is similarly updated with the z location of the new pixel (see Figure 47). Transparency blending may be accomplished by blending without updating the z buffer value--but nevertheless reading it first and not blending if the transparent pixel is behind an opaque pixel.

[Video Interface 210]

Video interface 210 reads the data out of frame buffer 118 and generates the composite, S video RGB video output signals. In this example, video interface 210 also performs anti-aliasing operations, and may also perform filtering to remove truncation caused by the introduction of dithering noise.

Video Interface 210 in this example works in either NTSC or PAL mode, and can display 15-bit or a 24-bit color pixels with or without filtering at both high and low resolutions. The video interface 210 can also scale up a smaller image to fill the screen. The video interface 210 provides 28 different video modes plus additional special features.

Video interface 210 reads color frame buffer 118a in synchronization with the electron beam scanning the color television screen 60, and provides RGB values for each pixel in digital form to video DAC 144 for conversion into analog video levels in this example. Video interface 210 performs a blending function for opacity values based on coverage (thereby providing an antialiasing function), and also performs a back-filtering operation to remove some of the noise introduced by screen-based dithering.

Figure 48 is a block diagram of the architecture of video Interface 210. In this example, video interface 210 includes the DMA controller 900, a buffer 902, control logic

整理番号=000568

(170)

904, anti-aliasing filters 906a, 906b, error correction blocks 908a, 908b, vertical interpolator (filter) 910, horizontal interpolator (filter) 912, "random" function generator 914, gamma block 916, and bus driver 918.

DMA controller 900 is connected coprocessor bus 214. DMA controller 900 reads color frame buffer 118a beginning at an "origin" address in the main memory specified by main process 100 (see Figure 50(b)). DMA controller 900 sequentially reads the pixel color and coverage values (see Figure 45) from frame buffer 118a in synchronism with the line scanning operations of television 58. The pixel values read by DMA controller 900 are processed by the remainder of video interface 210 and are outputted to video DAC 144 for conversion into an analog composite video signal NTSC or PAL format in this example.

DMA controller 900 in this example provides the color/coverage values it has read from main memory frame buffer 118a, to a RAM buffer 902 for temporary storage. In this example, buffer 902 does not store the pixel color values corresponding to an entire line of television video. Instead, buffer 902 stores a plurality of blocks of pixel data, each block corresponding to a portion of a line of video. Buffer 902 provides "double buffering," i.e., it has sufficient buffers to make some line portions available to filters 906 while other buffers are being written by DMA controller 900.

In this example, DMA controller 900 accesses, and stores into buffers 902, several of the pixel data corresponding to several horizontally-aligned portions of the video lines to be displayed on television screen 60. Looking at Figure 49, frame buffer 118a is shown -- for purposes of illustration -- as being organized in a row/column order corresponding to pixels on the television screen (it will be understood that the frame buffer as stored in main memory 300 may actually be stored as a long sequential list of pixel color/coverage values). In this example, DMA controller 900 reads out a block of pixel values corresponding to a particular segment of the current line n of video to be displayed (top shaded block in Figure 49 frame buffer 118a), and also reads out the pixel values corresponding to a horizontally-aligned (on the television screen) line segment of a "next" video line n+1 (i.e., the part of the pixel data representing the part of the next line

整理番号=000568

(171)

just beneath the line n). In this particular example, also reads a further block of pixel values from the frame buffer corresponding to the horizontally-aligned line segment of video line $n+2$.

Each of these blocks of pixel values is stored in buffer 902. Filters 906a, 906b perform a filtering/anti-aliasing operation based on coverage value to interpolate the current line's pixel values with neighboring pixel values (i.e., pixel values that are adjacent with respect to the displayed position on color television screen 60). The anti-aliasing filtering operations performed by filters 906a, 906b are as described in copending Japanese Patent Application Reference No. 0566 filed by Nintendo co., Ltd. and Silicon Graphics, Inc., entitled "Antialiasing of Silhouette Edges". Briefly, a three-scan-line high neighborhood is color weighted by coverage value in a blending process performed by filter 906. This filtering operation results in smoother, less jagged lines at surface edges by using the pixel coverage value retained in frame buffer 118a (which coverage value indicates what percentage of the pixel is covered by a polygon) to adjust the contribution of that pixel value relative to the contributions of neighboring pixel values in a blending process to produce the current pixel value. Divot error correction blocks 908a, 908b correct the outputs of anti-alias filters 906a, 906b for slight artifacts introduced by the anti-aliasing process. In particular, for any pixels on or adjacent to a silhouette edge, the error correction blocks 908 take the median of three adjacent pixels as the color to be displayed in place of the center pixel. This error correction can be enabled or disabled under software control (see Figure 50(a)), and a video game programmer may wish to disable the error correction since it interacts poorly with decal line rendering modes.

Anti-aliasing filters 906a, 906b operate in parallel in this example to produce pixel data blocks corresponding to horizontally aligned portions of two successive lines (line n , line $n+1$) of the image represented by frame buffer 118a. These pixel values are provided to vertical Interpolator 910, which performs a linear interpolation between the two image lines to produce an image portion of a single scan line (see Figure 49). Interpolator 910 interpolates between successive scan lines in order to reduce flicker in interlaced displays. For example, interpolator 910 can add in a contribution from a previous or next

整理番号=0 0 0 5 6 8

(172)

successive horizontally-aligned scan line portion to make transitions between successive video scan lines less noticeable -- thereby reducing flicker.

Additionally, interpolator 910 in this example can perform a vertical scaling function that allows the number of lines displayed on television screen 60 to be different from the number of lines represented by the frame buffer 118a pixel information. In this example, filter 906 scales in the vertical dimension by resampling the pixel data for successive lines of image represented by frame buffer 118a -- thereby allowing television screen 60 to have a different number of lines. This scaling operation (which also accommodates offsetting) is controlled by the values within the video interface Y scale register (see Figure 52(n)). The ability to scale the television image relative to the digital image size of frame buffer 118a provides additional flexibility. For example, the scaling ability makes it possible for signal processor 400 and display processor 500 to generate a smaller digital image representation in frame buffer 118 -- and yet allow that smaller image to fill the entire television screen 60. Since a smaller frame buffer 118 requires less time to rasterize (i.e., display processor 500 needs to handle fewer spans and fewer pixels per span for a given polygon) and less memory to store, the scaling ability can provide increased performance -- albeit at the cost of a lower resolution image.

The output of vertical filter 910 in this example is a block of pixel data representing the pixel values for a portion of the video line to be displayed. As shown in Figure 49, this block of pixel values is provided to horizontal interpolator 912. Horizontal interpolator 912 provides a linear interpolation between neighboring pixel values in order to resample the pixels based on a horizontal scaling factor stored in the X scale register (see Figure 52(m)). Horizontal interpolator 112 thus provides a horizontal scaling ability, e.g., to convert a smaller number of frame buffer values into a larger number of screen pixels along a horizontal line.

The output of horizontal interpolator 912 is provided to a Gamma correction circuit 916 that converts linear RGB intensity into non-linear intensity values suitable for composite video generation for the gamma non-linearity of TV monitors. This amounts to taking a square root of the linear color space. The TV monitor effectively raises these

整理番号=000568

(173)

color values to a power of 2.2 or 2.4. A "random" function block 914 introduces additional bits of resolution to each of the R, G and B color values in order to "de-dither" (i.e., to compensate for the bit truncation performed by display processor dithering block 522). As shown in Figure 45, one example frame buffer 118 color pixel format in this example provides only five bits of resolution of each R, G and B to conserve storage space within main memory 300. Display processor dithering block 522 may truncate 8-bit RGB color values provided by blender 510 to provide the compressed representation shown in Figure 45. Block 914 can reverse this truncation process to decompress the RGB values to provide 256 different display color levels for each R, G and B. See copending Japanese Patent Application Reference No. 0567 filed by Nintendo Co., Ltd. and Silicon Graphics, Inc., entitled "Restoration Filter For Truncated Pixels". This dither filter operation can be turned on and off under software control (see Figure 50(a)).

[Example Video Interface Registers]

There are sixteen control registers for the video interface 210 which control all its functions including sync generation, video rescaling, and anti-aliasing. Figures 50(a)-52(p) show the various registers within video interface 210 that can be accessed by main processor 100.

Figure 50(a) shows the video interface control register 952. Main processor 100 can write the following values into this register 952 to control the operation of video interface 210:

- Type field 952a specifies pixel data size as blank (no data, no sync), the format shown in Figure 45 (5bits each of RGB and a 3-bit coverage value), or 8/8/8/8 (32-bit color value and 8 bits of coverage);
- Gamma dither enable field 952b turns on and off the addition of some random noise to the least significant bits of the video out before the final quantization to 7 bits to eliminate Mach banding artifacts;
- Gamma enable field 952c turns on and off gamma correction;
- Divot enable field 952d turns on and off the divot error correction discussed above;

整理番号=000568

(174)

- vid o bus clock enable field 952e turns an internal clock on or off;
- Interlace field 952f turns interlacing on and off;
- Test mode field 952g;
- anti-alias mode on/off field 952h;
- diagnostic field 952i;
- pixel advance field 952j; and
- dither filter enable field 952k.

Figure 50(b) shows the video interface origin register 954 used to specify the beginning main memory 300 address of frame buffer 118a for read out. In this example, main processor 100 needs to explicitly set this register 954 each time video interface 210 is to read from a new area in main memory 300 (e.g., to read the other half of double buffered frame buffer 118).

Figure 50(c) shows the video interface line width register 956, which can be set to specify the number of pixels in each horizontal line. Figure 50(d) shows the video interface vertical interrupt register 958, which main processor 100 can set with a particular vertical line number so that coprocessor 200 will interrupt the main processor once per frame at the specified vertical line or half line. Figure 50(e) shows the video interface current line register 960, which specifies the current vertical line when read from by the main processor 100 and clears the vertical line interrupt when written to by the main processor.

The registers 962-972 shown in Figures 51(g)-52(l) are used by main processor to specify detailed composite video timing parameters. For example:

Figure 51(f) shows the vertical interface timing register 962 which main processor 100 can write to to specify horizontal sync pulse width, color burst width, vertical sync pulse width, and color burst start timing.

Figure 51(g) shows the video interface vertical sync register 964 that main processor 100 may write to specify the number of vertical half-lines per field.

Figure 51(h) shows the video interface horizontal sync register 965 which main processor 100 can write to specify the total duration of a line and a horizontal leap pattern for PAL.

Figure 51(i) shows the video interface h sync leap register 966 specifying two alternate h sync leap parameters for PAL.

The video interface horizontal video register and vertical video register 968, 970 shown in Figures 51(j), 51(k), respectively, are used to specify horizontal and vertical video start and end times relative to hsync and vsync.

The vertical interfaced vertical burst register 972 shown in Figure 52(l) specifies color burst start and end timing.

The timing parameters programmable into registers 962-972 can be used to provide compatibility with different kinds of television sets 58. For example, most television sets 58 in the United States use a composite video format known as NTSC, whereas most European television sets use a composite video format known as PAL. These formats differ in terms of their detailed timing parameters (e.g., vertical blanking integral width and location within the signal pattern, horizontal synchronization pulse width, color burst signal pulse width, etc.). Because registers 962-972 control these composite video timing parameters and are programmable by software executing on main processor 100, a programmer of video game 108 can make her program NTSC compatible, PAL compatible, or both (as selected by a user) by including appropriate instructions within the video game program that write appropriate values to registers 962-972. Thus, in this example, coprocessor 200 is compatible NTSC-standard television sets 58, PAL standard compatible television sets – and even with video formats other than these within a range as specified by the contents of registers 962-972.

整理番号=000568

(176)

Vertical interface x and y scale registers 974, 976 (see Figure 52(m), 52(n), respectively) specify x and y scale up and subpixel offset parameters for horizontal and vertical scaling, as discussed above. Figures 52(o) and 52(p) show video interface test data and address registers 978, 980 for diagnostic purposes.

[Memory Controller/Interface 212]

As explained above, coprocessor memory interface 212 interfaces main memory 300 with coprocessor internal bus 214. In this example, main memory 300 is accessed over a 9-bit wide bus, and one of the tasks memory interface 212 is responsible for is to buffer successive 9-bit words so they can be more conveniently handled by coprocessor 200. Figure 53 is an example diagram showing the overall architecture of memory controller/interface 212

In this example, memory interface/controller 212 includes a pair of registers/buffers 1000, 1002, a control block 1004, and a RAM controller block 212b. RAM controller block 212b comprise RAM control circuits designed and specified by Rambus Inc. for controlling main memory 300. Registers 1000, 1002 are used to latch outgoing and incoming data, respectively. Control block 1004 controls the operation of memory interface 212.

[Example Memory Controller/Interface Registers]

Figures 54(a)-55(h) show example control registers used by main processor 100 to control memory interface 212. Figure 54(a) shows a read/write mode register specifying operating mode and whether transmit or receive is active (1052). Figure 54(b) shows a configuration register 1054 that specifies current control input and current control enable. Figure 54(c) represents a current mode register 1056 that is write only, with any writes to this register updating the current control register. Figure 54(d) shows a select register 1058 used to select receive or transmit. Figure 54(e) shows a latency register 1060 used to specify DMA latency/overlap. Figure 54(f) shows a refresh register 1062 that specifies clean and dirty refresh delay, indicates the current refresh bank, indicates whether refresh is enabled, indicates whether refresh is optimized, and includes a field specifying refresh

整理番号=000568

(177)

multi-bank device. Figure 55(g) shows an error register which in a read mode indicates NACK, ACK and over-range errors, and when written to by main processor 100 clears all error bits. Figure 55(h) shows a bank status register 1066 which, when read from indicates valid and dirty bits of the current bank, and when written to clears valid and sets dirty bits of the current bank.

[CPU INTERFACE]

Figure 56 shows a block diagram of coprocessor CPU interface 202 in this example. CPU interface 202 comprises a FIFO buffer 1102 and a control block 1104. FIFO buffer 1102 provides bidirectional buffering between the CPU SysAD multiplexed address/data bus 102a and the coprocessor multiplexed address/data bus 214D. Control block 1104 receives addresses asserted by the main processor 100 and places them onto the coprocessor address bus 214C. Control block 1104 also receives interrupt signals from the other parts of coprocessor 200, and receives command control signals from the main processor 100 via SysCMD bus 102b.

[Example CPU Interface Registers]

Figures 57(a)-57(d) show the registers contained within CPU interface 303 in this example. Figure 57(a) shows a CPU interface status/control register 1152 that controls coprocessor 200 when main processor 100 writes to the register and indicates overall coprocessor status when the main processor reads from the register. Main processor 100 can write to register 1152 to specify initialization code length, set or clear initialization mode, set or clear internal coprocessor bus test mode, clear display processor 400 interrupt, and set or clear main memory register mode. When main processor 100 reads from this register 1152, it can determine initialization code length, initialization mode, internal coprocessor bus test mode, and whether the coprocessor is operating in the main memory register mode.

整理番号=000568

(178)

Figure 57(b) shows a version register 1154 that main processor 100 can read from to determine version information pertaining to various components within coprocessor 200.

Figure 57(c) shows an interrupt register 1156 that main processor 100 can read from to determine the source of an interrupt it has received from coprocessor 200. In this example, a single line connects between coprocessor 200 and main processor 100 is used for interrupt purposes. Upon receiving a coprocessor interrupt, main processor 100 can read interrupt register (which contains an interrupt vector) to ascertain what component within coprocessor 200 (i.e., signal processor 400, serial interface 204, audio interface 208, video interface 210, parallel interface 206, or display processor 500) cause the interrupt. Figure 57(d) shows an interrupt mask register 1158 which main processor 100 can write to to set or clear an interrupt mask for any of the interrupts specified in interrupt register 1156, and may read to determine interrupts are masked and which are not.

[AUDIO INTERFACE]

Figure 58 shows an overall block diagram architecture of audio interface 208 in this example. Audio interface 208 includes DMA logic 1200, a state machine/controller 1202, an audio clock generator 1204, audio data buffers 1206 and a serializer 1208. In this example, DMA logic 1200 fetches digital audio sample data from audio buffer 114 within main memory 300. DMA logic 1200 writes this audio sample data, 8 bytes at a time, into audio data buffers 1206. There are multiple audio data buffers 1206 arranged in a FIFO so that DMA logic 1200 can be prefetching some audio sample data while serializer 1208 serializes other, previously fetched-and-buffered audio sample data. Thus, buffers 1206 store enough data to supply serializer 1208 between block reads by DMA logic 1200. Since the output rate of serializer 1208 is relatively slow (e.g., on the order of 4 bytes at 50kHz, a single 64-bit buffer 1206b can store enough digitized audio samples to last a relatively long time in terms of real time audio output.

整理番号=000568

(179)

As discussed above, serializer converts the parallel contents of audio buffers 1206 into serial format, and places the resulting serial digital audio data stream onto bus 209 for communication to audio DAC 140. Digital audio bus 209 in this example includes a single serial data line 209a multiplexed between left channel data and right channel data. In this example, serializer 1208 outputs a 16-bit long word for each stereo channel, alternating between the channels. The output bit rate of serializer 1208 is specified by audio clock generator 1204. Audio clock generator 1204 produces an audio clock output on 209b to synchronize audio DAC 140 to the serializer 1208 output bit rate, and produces an audio L/R clock on line 209c specifying whether the current serializer output 1208 is for the left or right stereo channel.

Figure 58 shows a number of registers and counters used to control audio interface 208. DMA controllers 1200 receives a starting main memory address from an address register 1210. Main processor 100 writes to this address register 1210 (see Figure 59(a)) to point audio interface 208 to the locations in main memory 300 providing the audio buffer 114 for the current audio to be played. A counter 1212 increments this address for each fetch by DMA controller 1200 thereby sequencing the DMA controller through the entire audio buffer 114. Main process 100 writes the length of audio buffer 114 into a transfer length register 1214 (see Figure 59(b)). An additional counter 1216 associated with length register 1214 sequences state machine 1202 through an appropriate number of control states corresponding to the length of audio buffer 114. State machine 1202 generates control signals that synchronize the operations of the other parts of audio interface 208 relative to one another. In this example, main processor 100 can enable audio interface 208 to begin fetching data from the main memory 300 by writing to a DMA enable register location 1217 (not shown in Figure 58; see Figure 59(c)). Main processor 100 may also determine the state of audio interface 200 by reading an audio interface status register 1218 (not shown in Figure 58; see Figure 59(d)). In this example, state machine 1202 generates a main processor interrupt when it reaches the end of audio buffer 114 as specified by length register 1214, and the

整理番号=000568

(180)

main processor 100 can clear this interrupt by writing to the status register 1218 location (see Figure 59(d)).

In this example, main processor 100 may also control the rate of the clocking signals generated by audio clock generator 1204. Main processor 100 can program these rates by writing to audio rate registers 1218, 1220 (see Figures 59(e), 59(f)). A counter 1222 may provide a programmable dividing function based on the rate values main processor 100 as written into audio rate registers 1218, 1220.

[SERIAL INTERFACE]

Figure 60 shows an overall high level block diagram of serial interface 204 in this example.

In this example, serial interface 204 moves blocks of data between coprocessor 200 and serial peripheral interface 138. Serial interface 204 can either read a 64-byte data block from serial peripheral interface 138 and transfer it to a specified location in main memory 300 or alternatively, it can read a 64-byte block of data stored in the main memory and transfer it serially to the serial peripheral interface. In this example, serial interface 204 comprises primarily direct memory access logic 1300, control logic 1302, and a parallel/serial converter 1304. Parallel/serial converter 1304 In this example comprises a shift register that converts serial data sent by serial peripheral interface 138 over a read data/acknowledge bus 205a into parallel data for application to latch 1308. The contents of latch 1308 is then applied to coprocessor data bus 214d for writing into main memory 300. Alternatively, in a parallel-to-serial conversion mode, shift register 1304 receives parallel data from the coprocessor data bus 214d via a latch 1310 and converts that data into serial for transmission to serial peripheral interface 138 via a command and write data bus 205b.

Main processor 100 specifies the address within main memory 300 that serial interface 204 is to read from or write to, by writing this address into an address register 1312 (see Figure 61(a)). Address register 1312 contents specify the main memory address to be loaded in DMA address counter 1314. Part of the contents of address

整理番号= 0 0 0 5 6 8

(181)

register 1312 may also be used to specify "address" information within serial peripheral interface 138. Such serial peripheral interface "address" information is loaded into a latch 1316, the contents of which are provided to shift register 1304 for transmission to the serial peripheral interface. This serial peripheral interface "address" information may be used, for example, to specify a location within the serial peripheral interface 138 (i.e., a boot ROM location 158, a RAM buffer or a status register).

In this example, serial interface 204 has the ability to place the shift register 1304 parallel output onto the coprocessor address bus 214c via register 1308, a multiplexer 1318, and a latch 1320.

As shown in Figures 61(b), 61(c), main processor 100 in this example specifies the direction of serial transfer by writing to a location 1322 or 1324. A write to location 1322 causes serial interface 204 to read a 64-byte data block from the serial peripheral interface 138 and write it to the main memory 300 location specified by address register 1312. A write by main processor 100 to register location 1324 causes serial interface 204 to read a 64-byte block of data from the main memory 300 location specified by address register 1312, and to write the data in serial form to the serial peripheral interface 138.

Figure 61(d) shows the serial interface status register 1326. Main processor 100 can read status register 1326 to determine the status of serial interface 204 (e.g., whether the serial interface is busy with a DMA or I/O operation (fields 1328 (1) 1328 (2), respectively); whether there has been a DMA error (field 1328 (3)); or whether the serial interface has caused a main processor interrupt (field 1328 (4)). Serial interface 204 may generate a main processor interrupt each time it has completed a data transfer to/from serial peripheral interface 138. Main processor 100 can clear the serial interface interrupt by writing to register 1326.

[PARALLEL PERIPHERAL INTERFACE]

Figure 62 shows an example block diagram of parallel peripheral interface 206. In this example, parallel interface 206 transfers blocks of data between main memory 300 and storage device 54. Although storage device 54 described above includes only a

整理番号=0 0 0 5 6 8

(182)

read-only memory 76 connected to parallel bus 104, system 50 can accommodate different configurations of peripherals for connection to connector 154. For example, two different types of peripheral devices (e.g., a ROM and a RAM) may be connected to peripheral connector 154. Peripheral interface 206 is designed to support communications between two different types of peripheral devices connected to the same parallel bus 104 without requiring any time-consuming reconfiguration between writes.

Some such peripheral devices may be read-only (e.g., ROM 76), other such peripheral devices may be read/write (e.g., a random access memory or a modem), and still other such peripheral devices could be write only. Peripheral interface 206 supports bi-directional, parallel transfer over parallel bus 104 between connector 154 and main memory 300.

Parallel peripheral interface 206 in this example includes a DAM controller 1400, a control/register block 1402, and a register file 1404. Register file 1404 buffers blocks of data being transferred by peripheral interface 206 between a peripheral device connected to connector 154 and a block of storage locations within main memory 300. In this example, register file 1404 comprises a small RAM that stores 16 64-bit words. Register file 1404 operates as a FIFO, and is addressed by control/register block 1402. The output of register file 1404 is multiplexed into 16-bit portions by multiplexer 1406. These 16-bit-wide values are latched by a latch 1408 for application to the peripheral device connected to connector 154 via a multiplexed address/data bus 104ad. Data read from the peripheral device via the multiplexed address/data bus 104ad is temporarily stored in a latch 1410 before being applied (via a multiplexer 1412 that also positions the 16-bit read value within an appropriate quarter of a 64-bit word) into register file 1404. Multiplexer 1412 also receives data from coprocessor data bus 214d via latch 1414, and can route this received data into register file 1404 for storage. The register file 1404 output can also be coupled to coprocessor data bus 214d via latch 1416. In this example, the register file 1404 output may also be coupled to the coprocessor address bus 214c via a multiplexer 1418 and a latch 1420.

整理番号=000568

(183)

Main processor 100 controls the parameters of a DAM transfer performed by peripheral interface 206 by writing parameters into control/register block 1402. For example, main processor 100 can write a starting main memory address into a DRAM address register 1422 (see Figure 63(a) and can write a starting address space of a peripheral device connected to connector 154 by writing a peripheral bus address starting address into the peripheral bus register 1424 (see Figure 63(b)). In this example, main processor 100 specifies the length and direction of transfer by writing to one of registers 1426, 1428 shown in Figures 63(c), 63(d), respectively. A write to read length register 1426 shown in Figure 63(c) controls the peripheral interface 206 to transfer in one direction, whereas writing a length value into register 1428 shown in Figure 63(d) causes the peripheral interface to transfer in the opposite direction. In this example, the main processor 100 can read the status of peripheral interface 206 by reading from a status register location 1430(R) (See Figure 63(b)). This status register 1430(R) contains fields 1432 indicating DMA transfer in progress (field 1432 (1)), I/O operation in process (field 1432 (#)), an error condition (field 1432 (3)). By writing to the same register 1430(W) location, main processor 100 can clear an interrupt peripheral interface 206 generates when it has completed a requested transfer. Writing to status register location 1430(W) also allows main processor 100 to both clear and interrupt and abort a transfer in progress (see Figure 63(a) field 1434 (1)).

Figures 64(f), 64(g), 64(h), 64(i) show additional registers main processor 100 can write to in order to control timing and other parameters of the peripheral interface bus 104. These registers permit main processor 100 to configure the bus 104 for particular types of peripheral devices--all under control of software within game program 108. In this example, peripheral Interface 44 supports duplicate sets of registers 1436, 1438, 1440 and 1442 shown in Figures 64(f)-64(i)--allowing different peripheral bus 104 protocols to be used for different peripheral devices connected simultaneously to the bus without requiring the main processor 100 to re-write the configuration registers each time it request access to a different device. In this example, one set of configuration registers 1436, 1438, 1440 and 1442 are used to configure the bus 104 protocol whenever the

整理番号=000568

(184)

peripheral interface 206 accesses a region 1 address space within the 16-bit peripheral address space, in the other set of register parameters are used whenever the peripheral interface accesses a region 2 address space within the peripheral bus address range (see Figure 10 memory map). The configurations specified by these two sets of registers are invoked simply by main processor 100 writing to the appropriate region.

The various ones of control registers shown in Figures 63(a)-64(i) may, in this example, be located within the control/register block 1402 of Figure 62. The configuration values stored in registers 1436, 1438, 1442 are used in this example to control the timing of the access control signals control/register block 1402 produces on bus control line 1404C. A latch 1434 is used to temporarily latch addresses on the co-processor address bus 214C for application to control/register block 1402 (e.g., to select between the various registers). Control/register block 1402 in this example includes appropriate counters and the like to automatically increment DMA addresses.

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiment, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the spirit and scope of the above mentioned claims.

[BRIEF DESCRIPTION OF DRAWINGS]

Figure 1 shows an overall video game system capable of generating 3-D images and digitally processed stereo sound;

Figure 2 shows example 3-D screen effects achievable using the Figure 1 system;

Figure 3 shows example 3-D screen effects achievable using the Figure 1 system;

Figure 4 shows an example of principal components of an overall video game system;

Figure 5 shows example major processing operations of an overall video game system;

整理番号=000568

(185)

Figure 6 shows example overall operation of a video game system;

Figure 7 shows example overall steps performed by a video game system to generate graphics images;

Figure 8 shows a detailed overall system architecture example;

Figure 9 shows an example main processor initialization routine;

Figure 10 shows an example main processor memory map;

Figure 11 shows an example coprocessor internal architecture;

Figure 12 shows an example coprocessor internal bus architecture;

Figure 13 shows an example signal processor internal architecture;

Figure 14(a) shows an example signal processor instruction format;

Figure 14(b) shows an example slicing of the Figure 14(a) source or destination field for processing by the vector unit shown in Figure 13;

Figure 14(c) shows an example add operation performed by the example signal processor vector unit;

Figure 15 shows example signal processor registers;

Figure 16 shows example signal processor registers;

Figure 17 shows an example hierarchical task list including graphics display lists and audio play lists;

Figure 18 shows an example microcode load routine;

Figure 19 shows an example simple signal processor display list processing example;

Figure 20 shows an example signal processor graphics microcode control step sequence;

Figure 21(a) shows an example double precision representation;

Figure 21(b) shows an example matrix format;

Figure 22(a) shows an example signal processor vertex buffer format;

Figure 22(b) shows an example vertex data definition;

Figure 23 shows an example signal processor segment addressing arrangement;

Figure 24 shows an example audio software architecture;

整理番号=000568

(186)

Figure 25 shows an example simple signal processor play list processing example;

Figure 26 shows an example signal processor audio microcode control step sequence;

Figure 27 shows an example signal processor audio processing construct;

Figure 28 shows example overall display processor processing steps;

Figure 29 shows example display processor pipeline configurations;

Figure 30 shows an example display processor architecture;

Figure 31 shows example display processor registers;

Figure 32 shows example display processor registers;

Figure 33 shows example display processor registers;

Figure 34 shows example display processor registers;

Figure 35 shows an example texture memory tile descriptor arrangement;

Figure 36 shows an example texture unit process;

Figure 37 shows an example texture coordinate unit and texture memory unit architecture;

Figure 38 shows an example texture memory color index mode lookup;

Figure 39 shows an example more detailed use of the texture memory to store color indexed textures;

Figure 40 shows an example color combiner operation;

Figure 41 shows an example alpha combiner operation;

Figure 42 shows an example alpha fix up operation;

Figure 43 shows an example of blending different types of primitives;

Figure 44 shows an example blender operation;

Figure 45 shows an example color pixel format;

Figure 46 shows an example depth (z) pixel format;

Figure 47 shows an example write enable generation process;

Figure 48 shows an example video interface architecture;

Figure 49 shows an example video interface operating sequence;

Figure 50 shows example video interface control registers;

Figure 51 shows example video interface control registers;

Figure 52 shows example video interface control registers;

Figure 53 shows an example main memory interface architecture;

Figure 54 shows example memory interface controller registers;

Figure 55 shows example memory interface controller registers;

Figure 56 shows an example main processor interface architecture;

Figure 57 show example main processor interface registers;

Figure 58 shows an example audio interface architecture;

Figure 59 shows example audio interface registers;

Figure 60 shows an example serial interface architecture;

Figure 61 shows example serial interface registers;

Figure 62 shows an example peripheral interface architecture;

Figure 63 shows example peripheral interface control/status registers;

Figure 64 shows example peripheral interface control/status registers;

Figure 65 shows formats and associated functions of a graphis display command that is "Set Color Image" of a display processor;

Figures 66 shows formats and associated functions of a graphis display command that is "Set Texture Image" of a display processor;

Figure 67 shows formats and associated functions of a graphis display command that is "Set Z Image" of a display processor;

Figure 68 shows formats and associated functions of a graphis display command that is "Set Tile" of a display processor;

Figure 69 shows formats and associated functions of a graphis display command that is "Load Tile" of a display processor;

Figure 70 shows formats and associated functions of a graphis display command that is "Load Block" of a display processor;

Figure 71 shows formats and associated functions of a graphis display command that is "Set Tile Size" of a display processor;

整理番号=000568(188)

Figure 72 shows formats and associated functions of a graphis display command that is "Load Tlut" of a display processor;

Figure 73 shows Triangle command of various types;

Figure 74 shows formats and associated functions of a graphis display command that is "Edge Coefficients" of a display processor;

Figure 75 shows formats and associated functions of a graphis display command that is "Edge Coefficients" of a display processor;

Figure 76 shows formats and associated functions of a graphis display command that is "Edge Coefficients" of a display processor;

Figure 77 shows formats and associated functions of a graphis display command that is "Shade Coefficients" of a display processor;

Figure 78 shows formats and associated functions of a graphis display command that is "Shade Coefficients" of a display processor;

Figure 79 shows formats and associated functions of a graphis display command that is "Texture Coefficients" of a display processor;

Figure 80 shows formats and associated functions of a graphis display command that is "Texture Coefficients" of a display processor;

Figure 81 shows formats and associated functions of a graphis display command that is "Z Buffer Coefficients" of a display processor;

Figure 82 shows formats and associated functions of a graphis display command that is "Fill Rectangle" of a display processor;

Figure 83 shows formats and associated functions of a graphis display command that is "Texture Rectangle" of a display processor;

Figure 84 shows formats and associated functions of a graphis display command that is "Texture Rectangle Flip" of a display processor;

Figure 85 shows formats and associated functions of a graphis display command that is "Set Combine Mode" of a display processor;

Figure 86 shows formats and associated functions of a graphis display command that is "Set Other Modes" of a display processor;

整理番号=000568

(189)

Figure 87 shows formats and associated functions of a graphis display command that is "Set Other Modes" of a display processor;

Figure 88 shows formats and associated functions of a graphis display command that is "Set Env Color" of a display processor;

Figure 89 shows formats and associated functions of a graphis display command that is "Set Prim Color" of a display processor;

Figure 90 shows formats and associated functions of a graphis display command that is "Set Blend Color" of a display processor;

Figure 91 shows formats and associated functions of a graphis display command that is "Set Fog Color" of a display processor;

Figure 92 shows formats and associated functions of a graphis display command that is "Set Fill Color" of a display processor;

Figure 93 shows formats and associated functions of a graphis display command that is "Set Prim Depth" of a display processor;

Figure 94 shows formats and associated functions of a graphis display command that is "Set Scissor" of a display processor;

Figure 95 shows formats and associated functions of a graphis display command that is "Set Convert" of a display processor;

Figure 96 shows formats and associated functions of a graphis display command that is "Set Key R" of a display processor;

Figure 97 shows formats and associated functions of a graphis display command that is "Set Key GB" of a display processor;

Figure 98 shows formats and associated functions of a graphis display command that is "Sync Full" of a display processor;

Figure 99 shows formats and associated functions of a graphis display command that is "Sync Load" of a display processor;

Figure 100 shows formats and associated functions of a graphis display command that is "Sync Pipe" of a display processor;

整理番号=000568(190)

Figur. 101 shows formats and associated functions of a graphis display command that is "Sync Tile" of a display processor; and

Figure 102 shows formats and associated functions of a graphis display command that is "No Op" of a display processor.

[BRIEF DESCRIPTION OF REFERENCE CHARACTERS]

52...Main Unit

100...Main Processor

200...Coprocessor

300...Main Memory

400...Signal Processor

500...Display Processor

FIG. 1

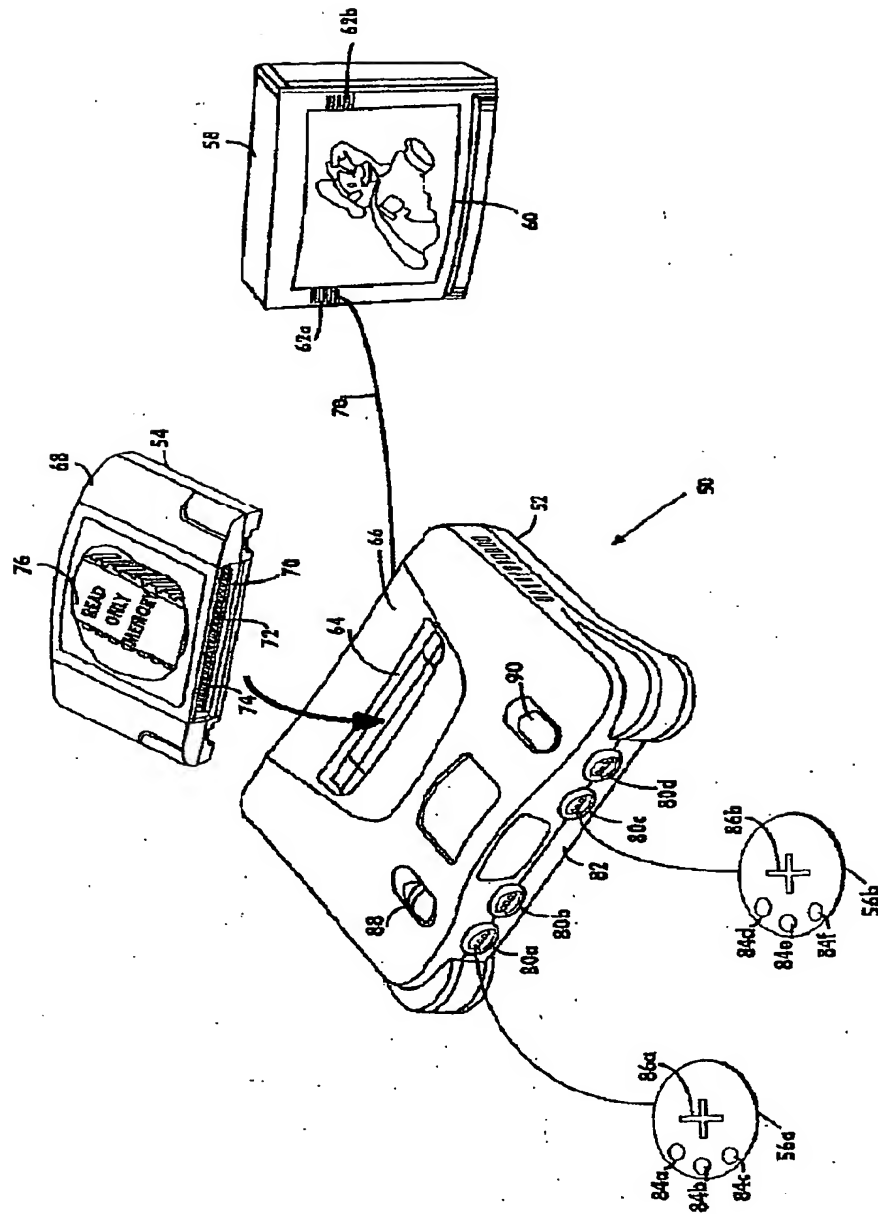
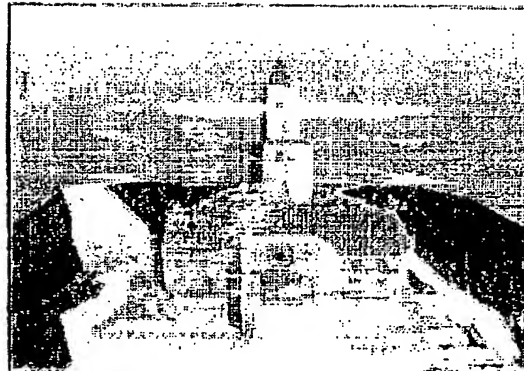


FIG. 2
(a)

(b)



(c)

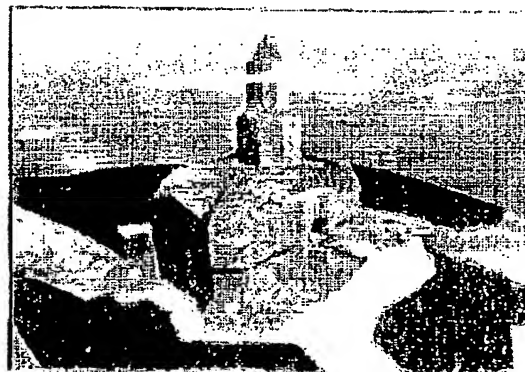
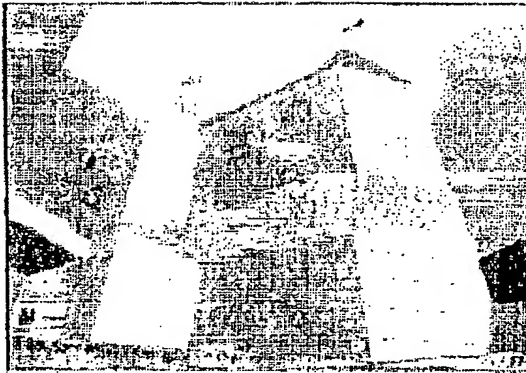
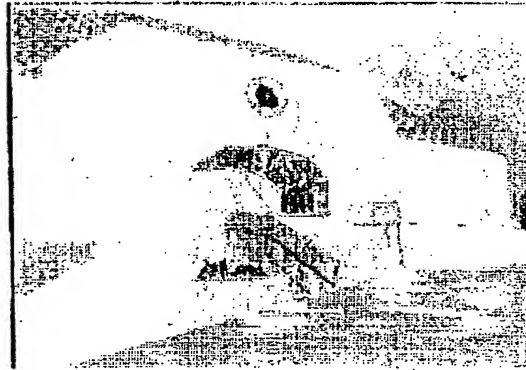


FIG. 3

(d)



(e)



(f)

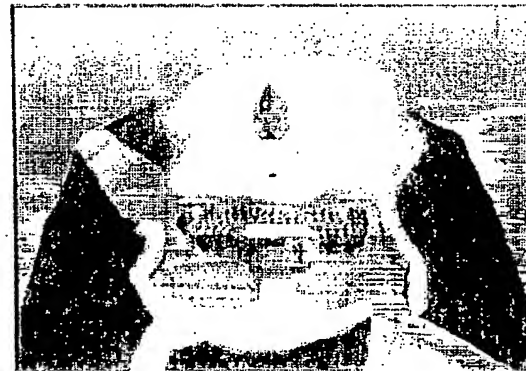


FIG. 4

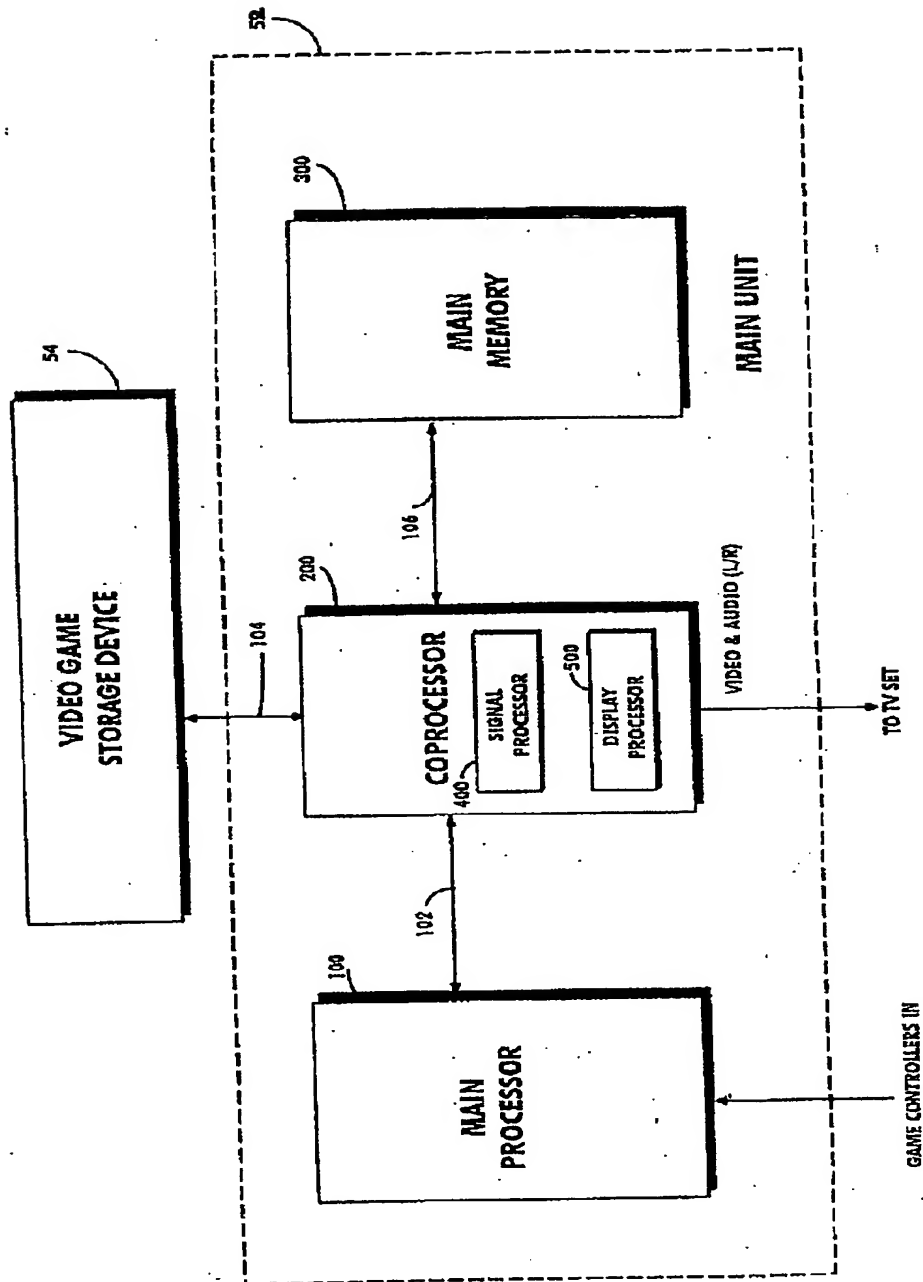


FIG. 5

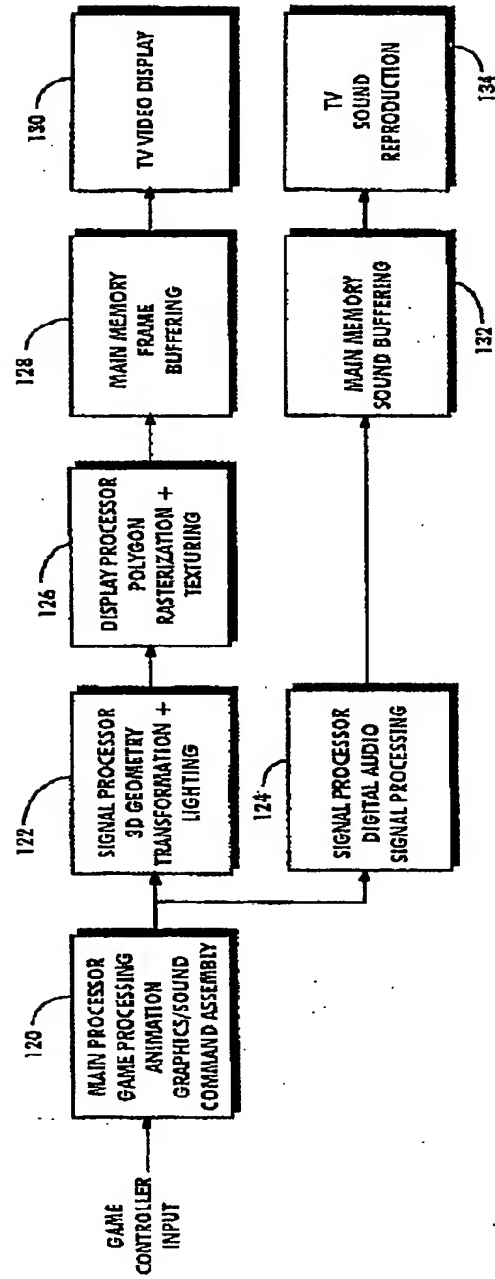


FIG. 6

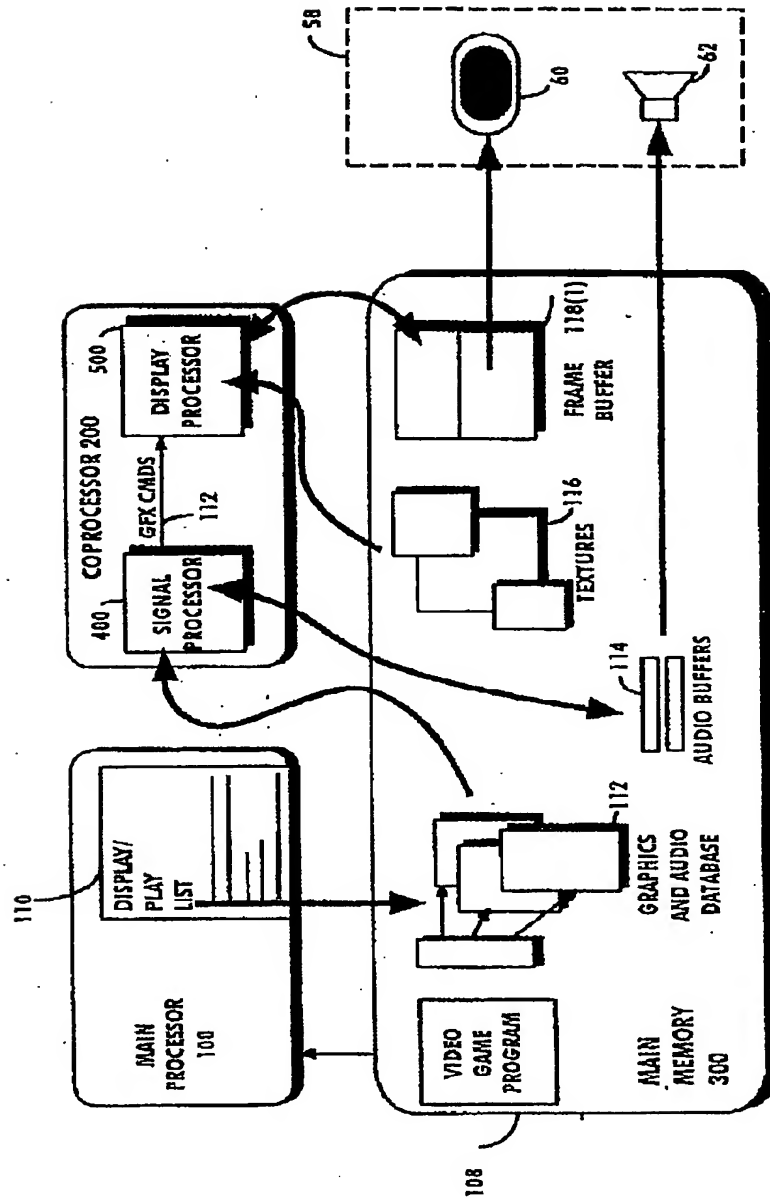


FIG. 7

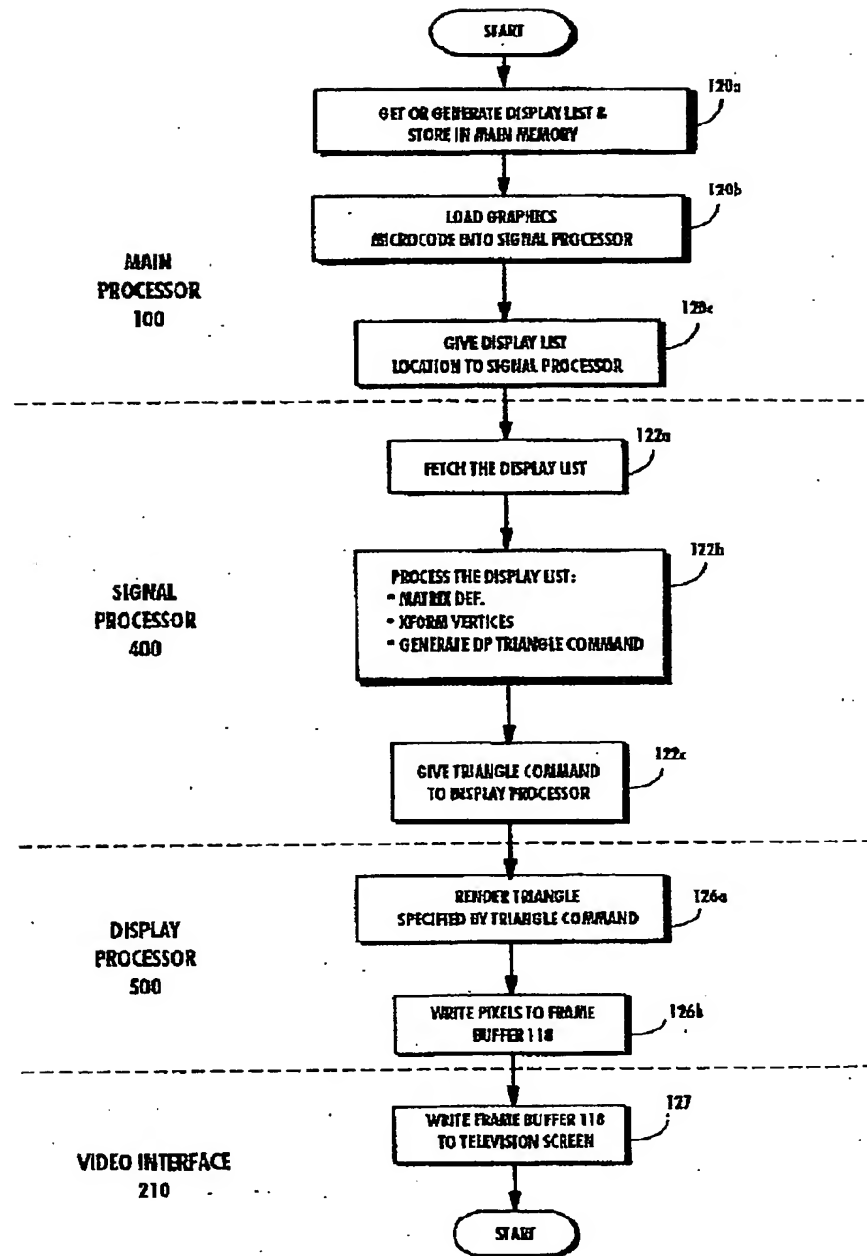


FIG. 8

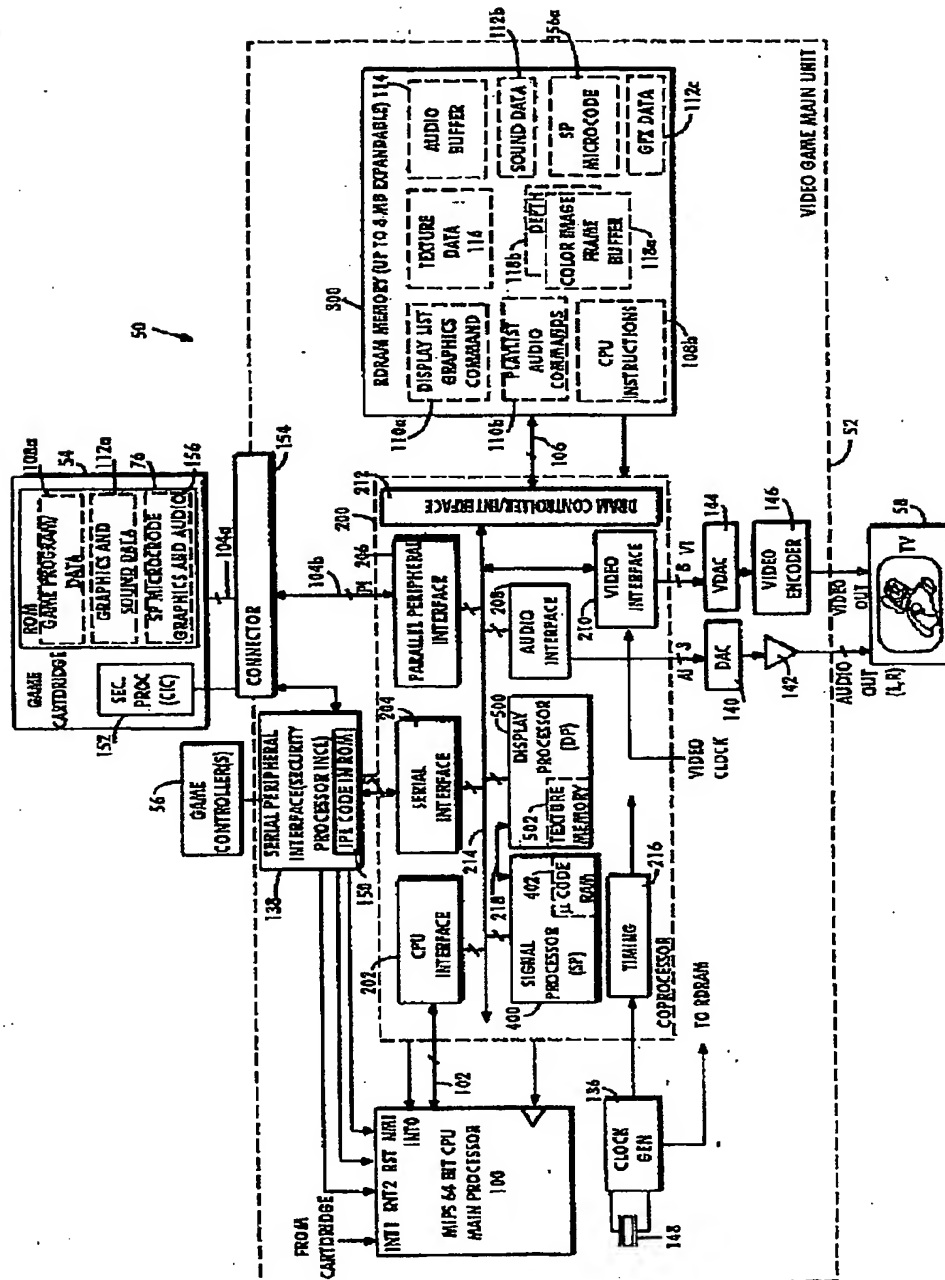


FIG. 9

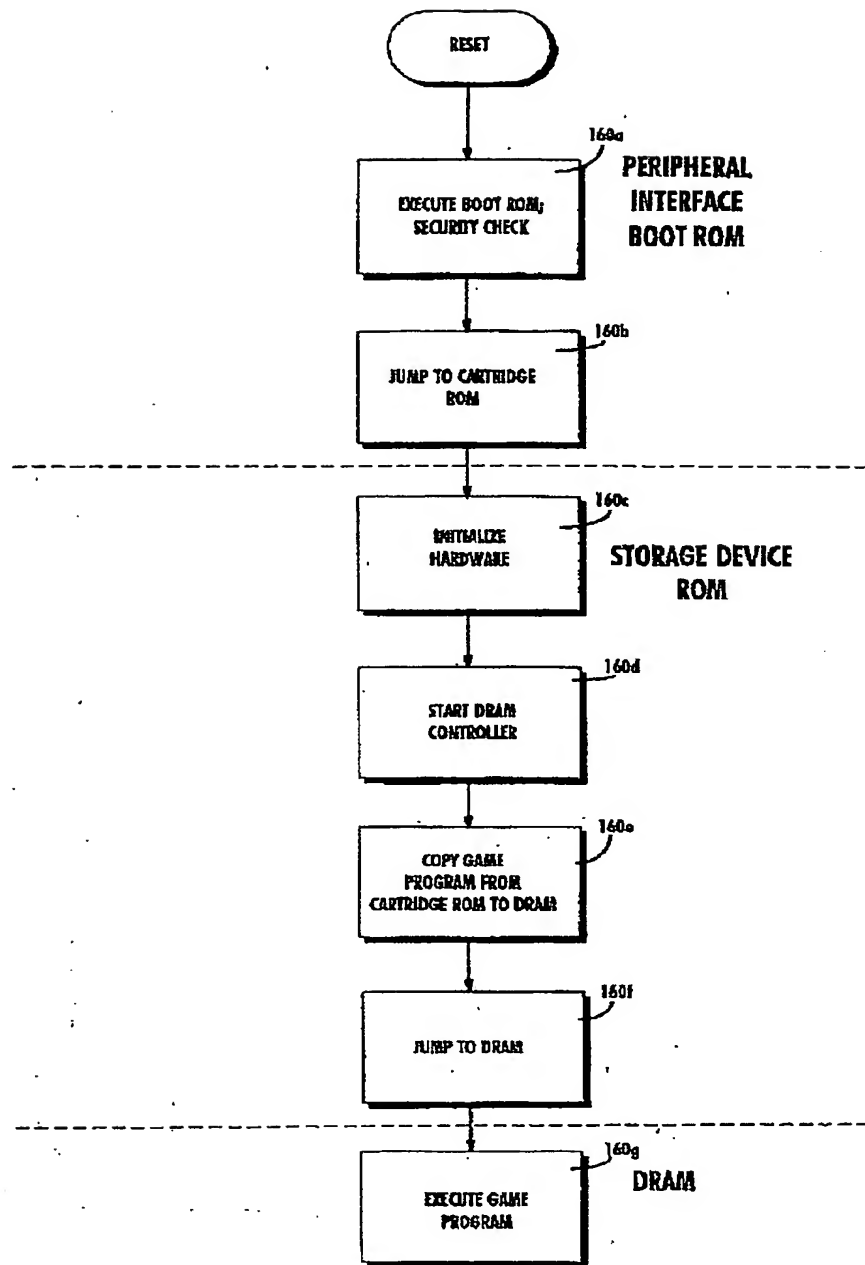


FIG. 10

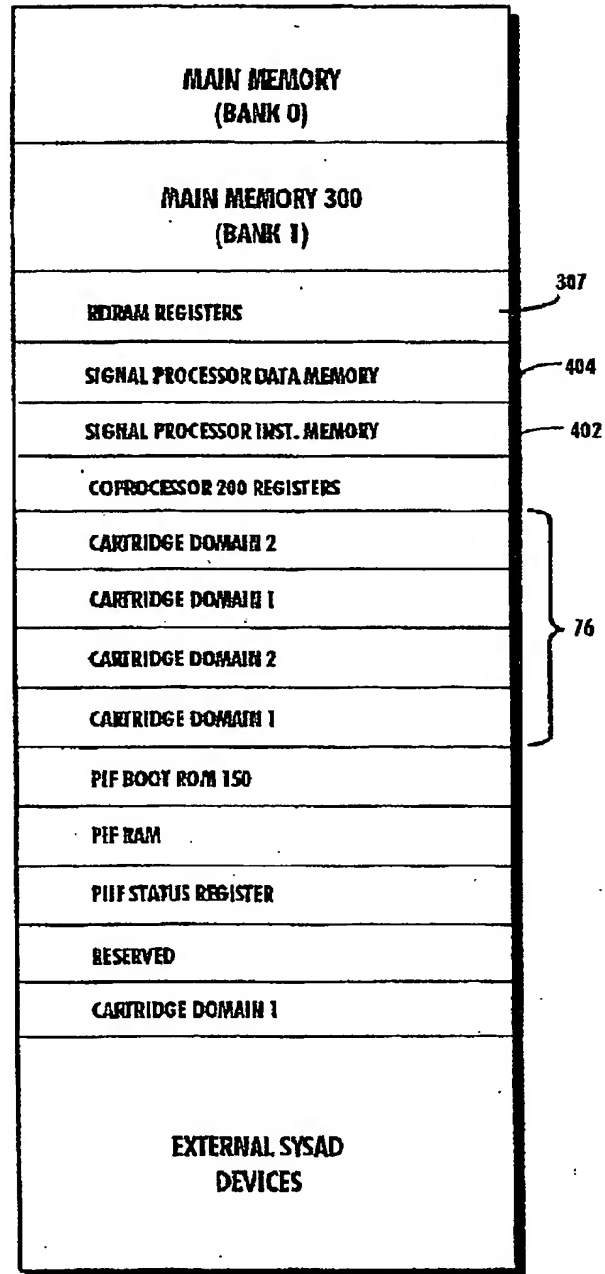


FIG. 11

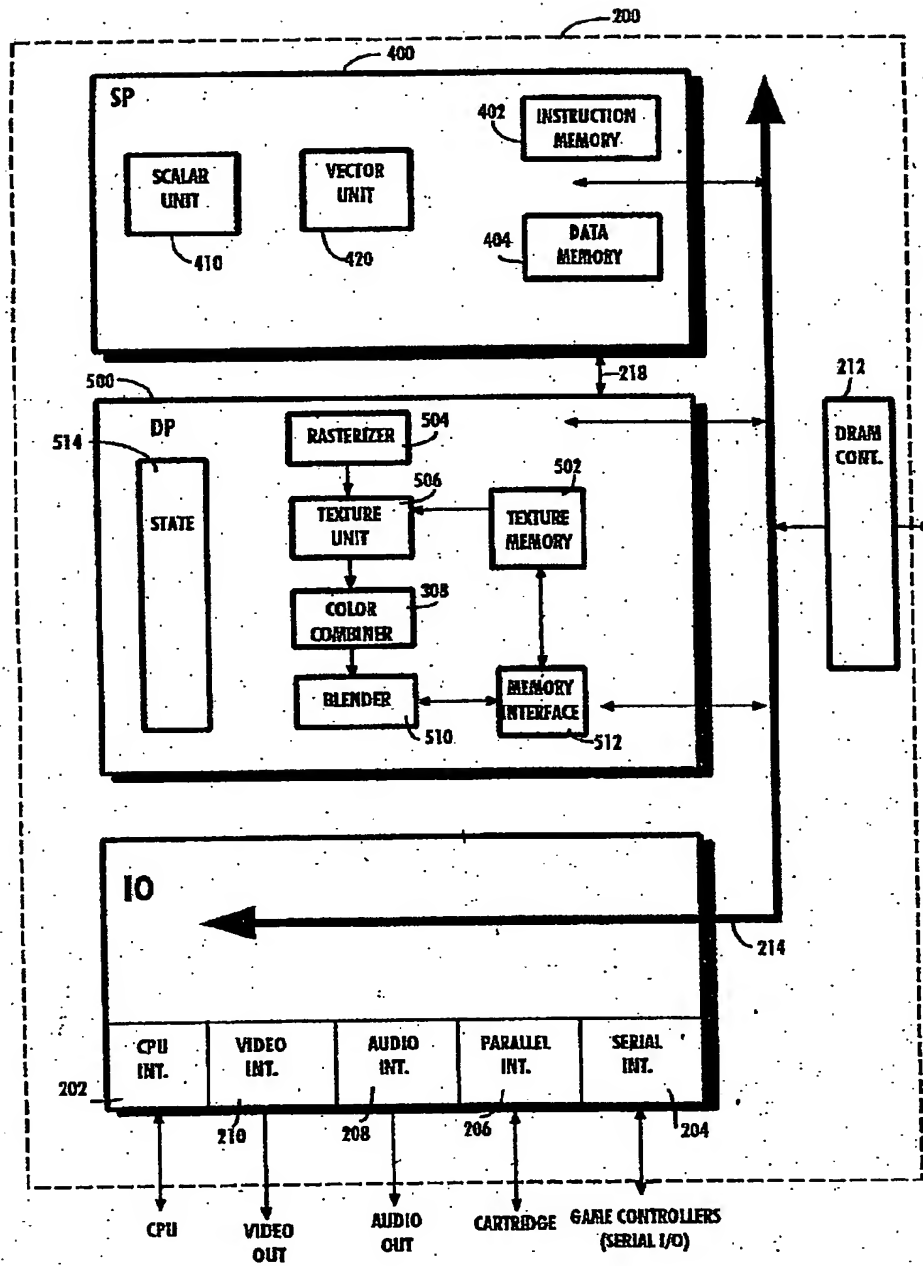


FIG. 12

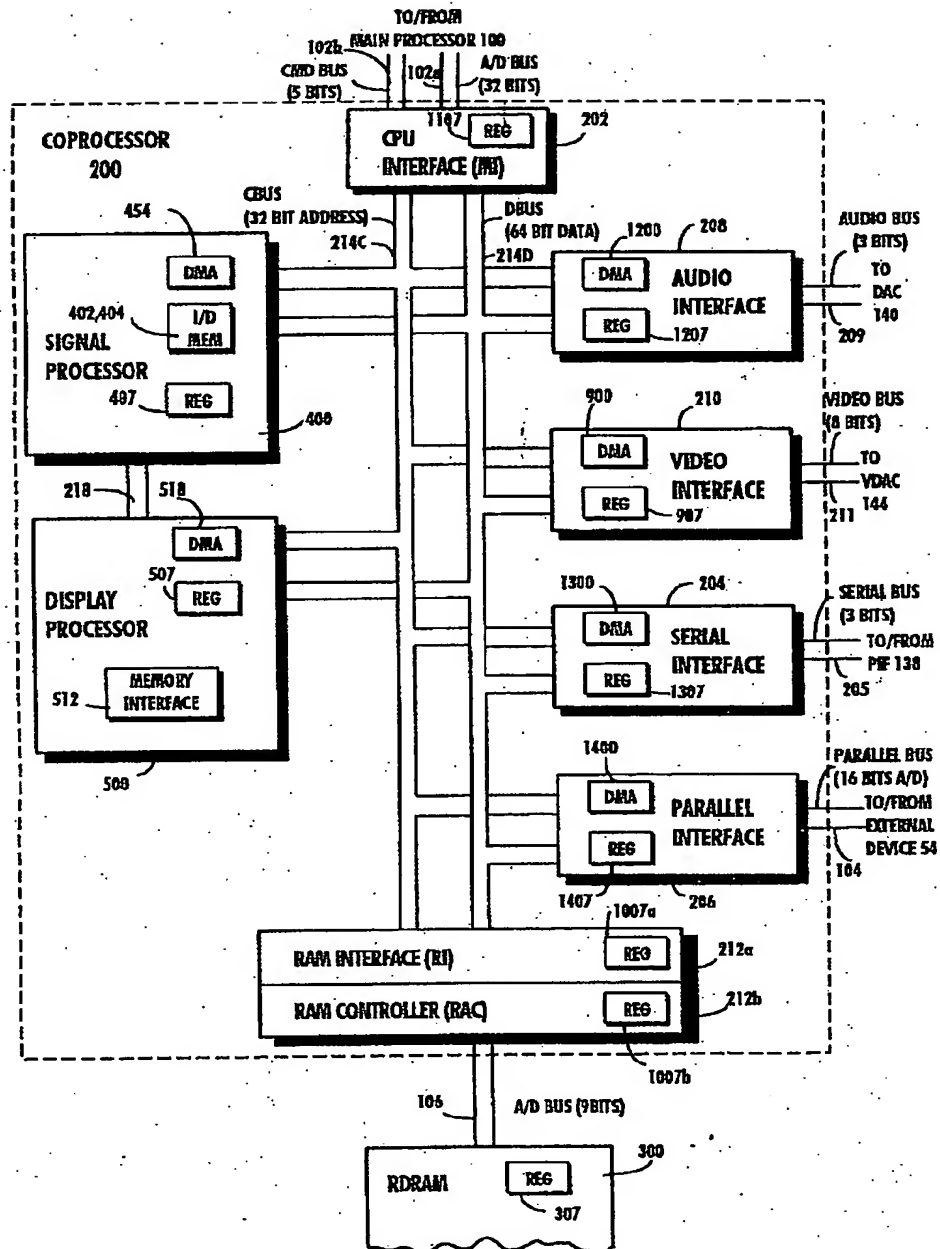


FIG. 13

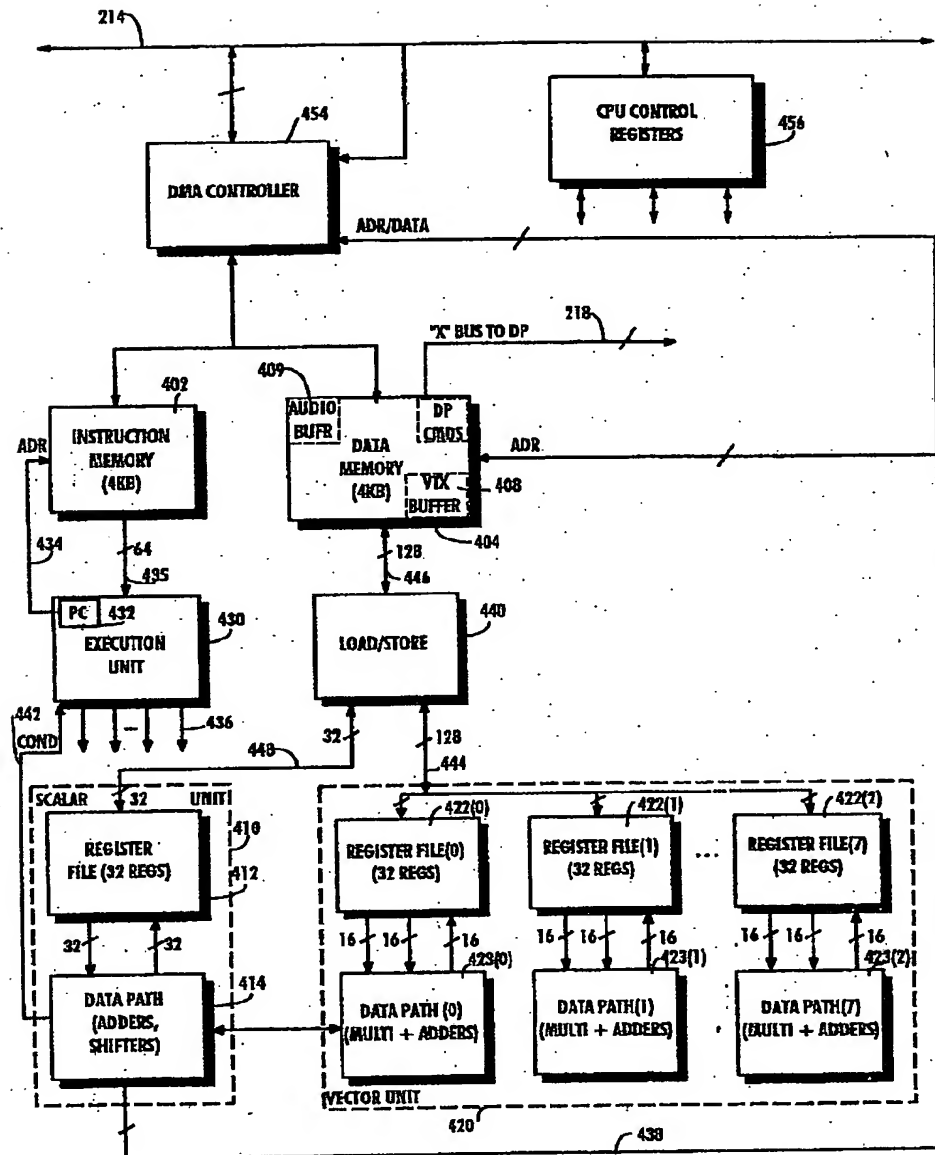


FIG. 14

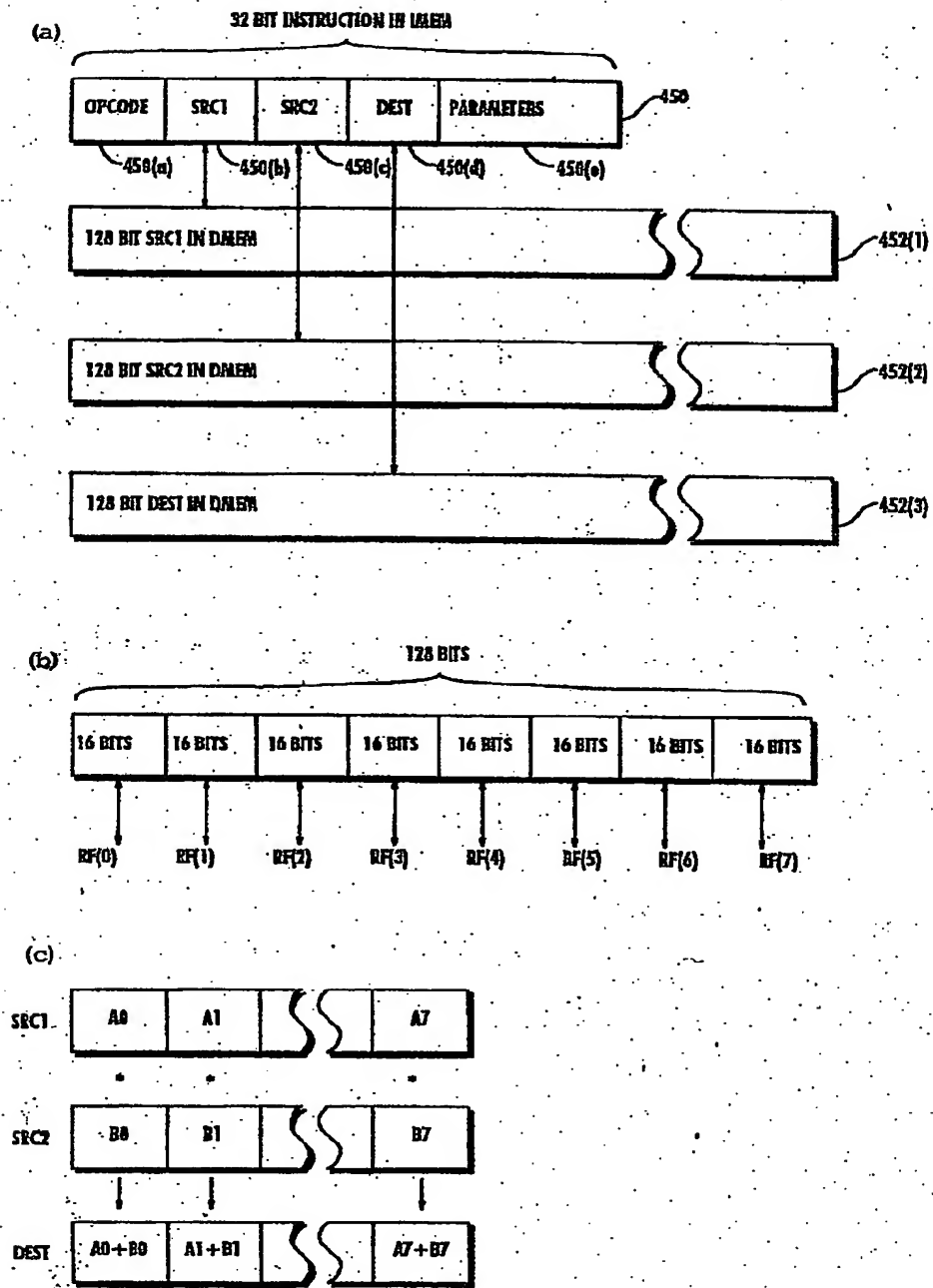
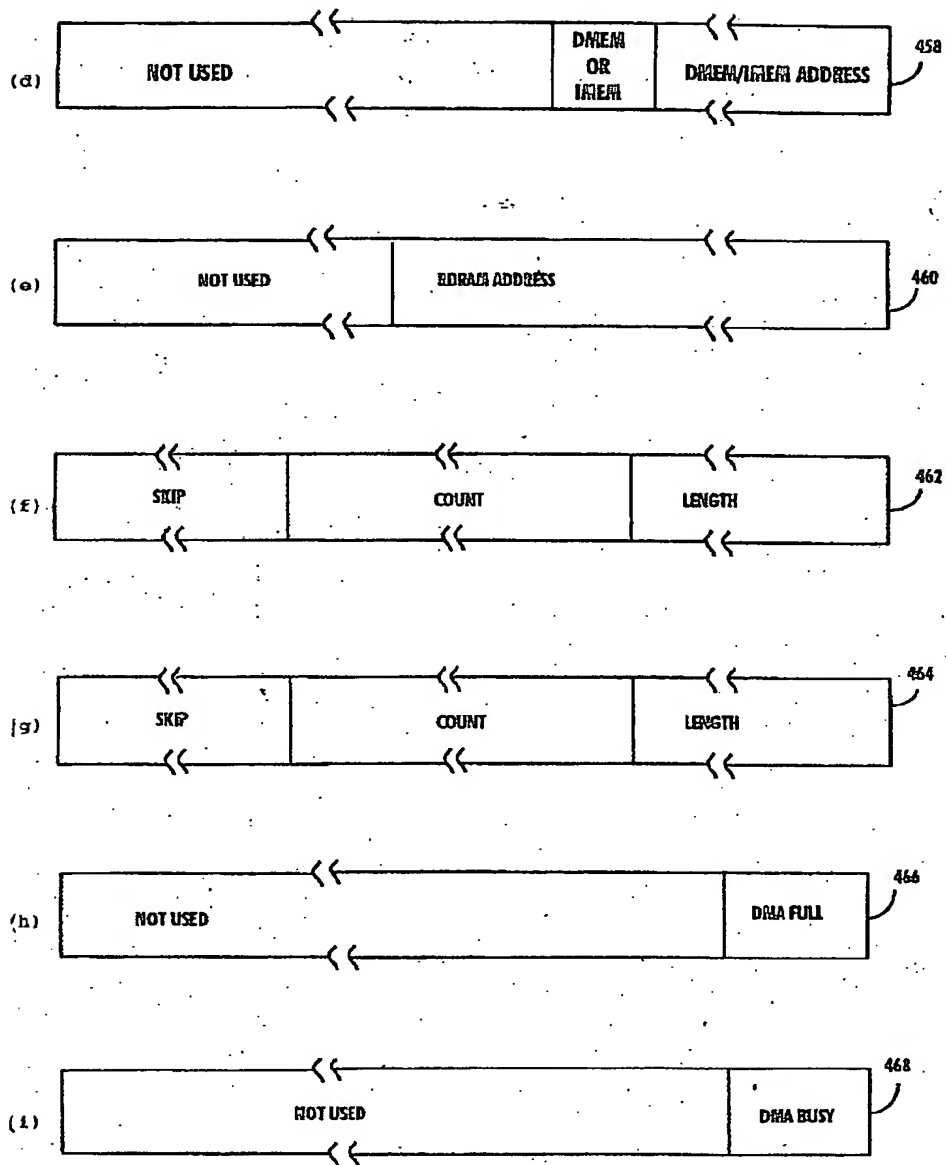


FIG. 15



(1)

489(1)		488(1)		489(0)		488(0)		487		486		485		484		483		482		481		480	
SET	SIG	CLR	SIG	SET	SIG	CLR	SIG	SET	INT	CLR	INT	SET	SS	CLR	SS	SET	INT	CLR	INT	SET	ERR	CLR	ERR
7	7	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

479(1) 479(0) 478 477 476 475 474 473 472 471 470(N)

479(1)		479(0)		478		477		476		475		474		473		472		471		470(N)	
SIG	7	SIG	0	INT	ON	SINGLE	I/O	DBA	FULL	DBA	BUSY	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
7	7	1	0	ON	ERR	STEP	FULL	FULL	FULL	BUSY	BUSY	ERR	ERR	ERR	ERR	ERR	ERR	ERR	

(2)

491	
NOT USED	FLAG (SET ON READ)

(3)

492(N)	
NOT USED	BIST CHECK

(4)

492(E)	
NOT USED	BIST CHECK

FIG. 17

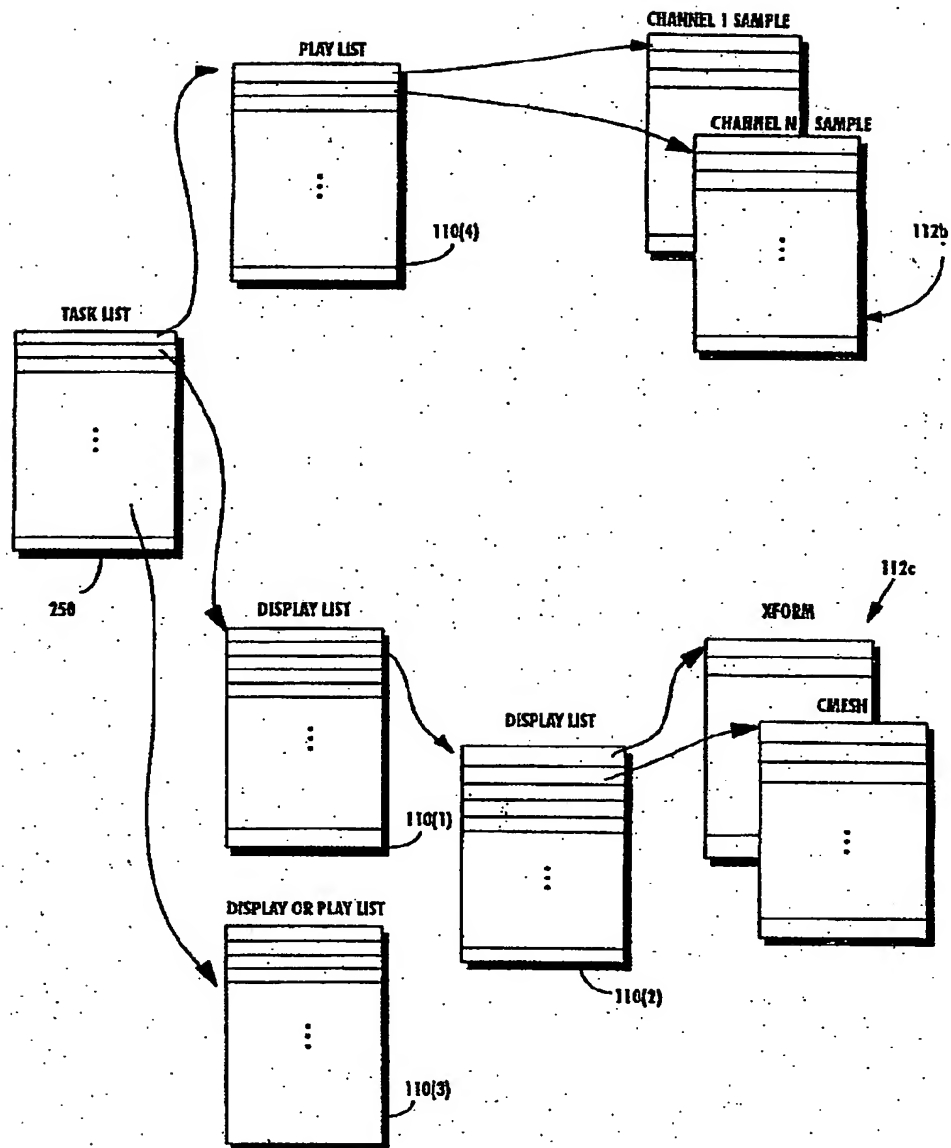


FIG. 18

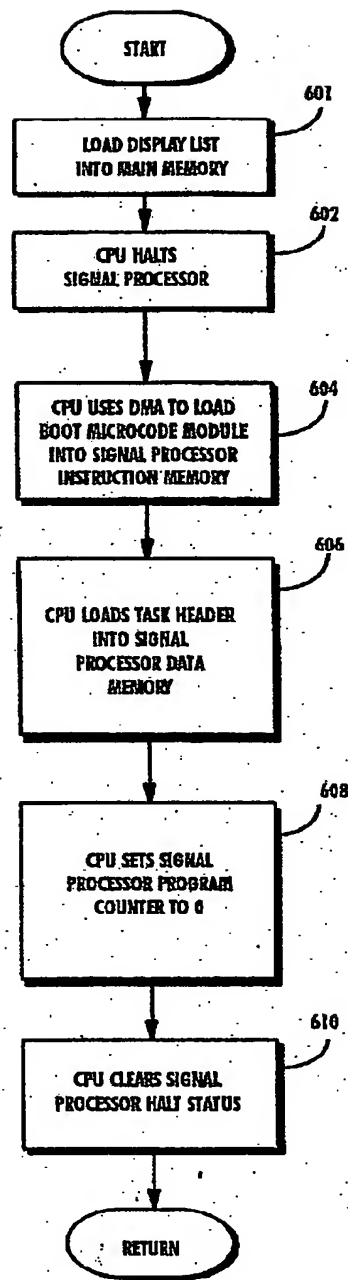


FIG. 19

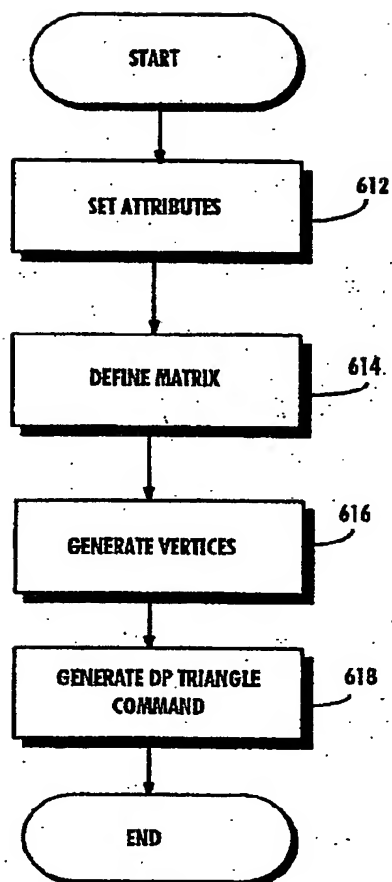


FIG. 20

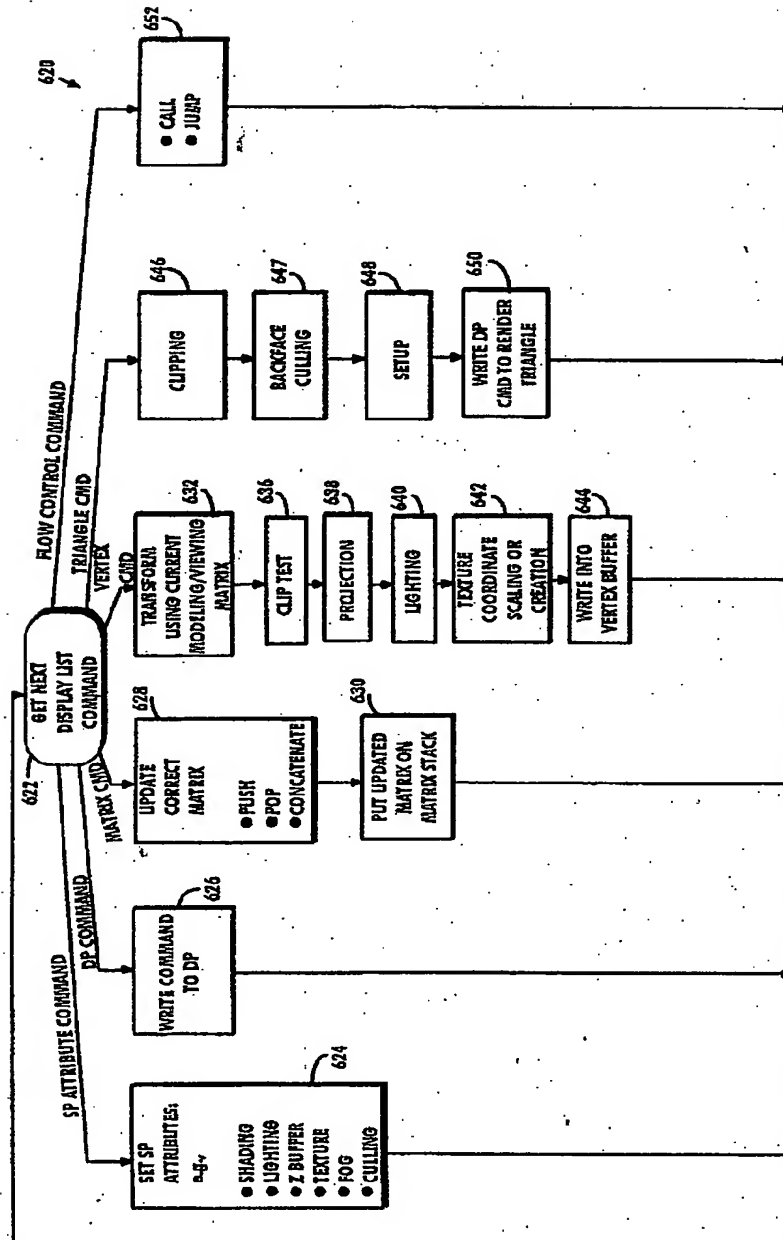


FIG. 21

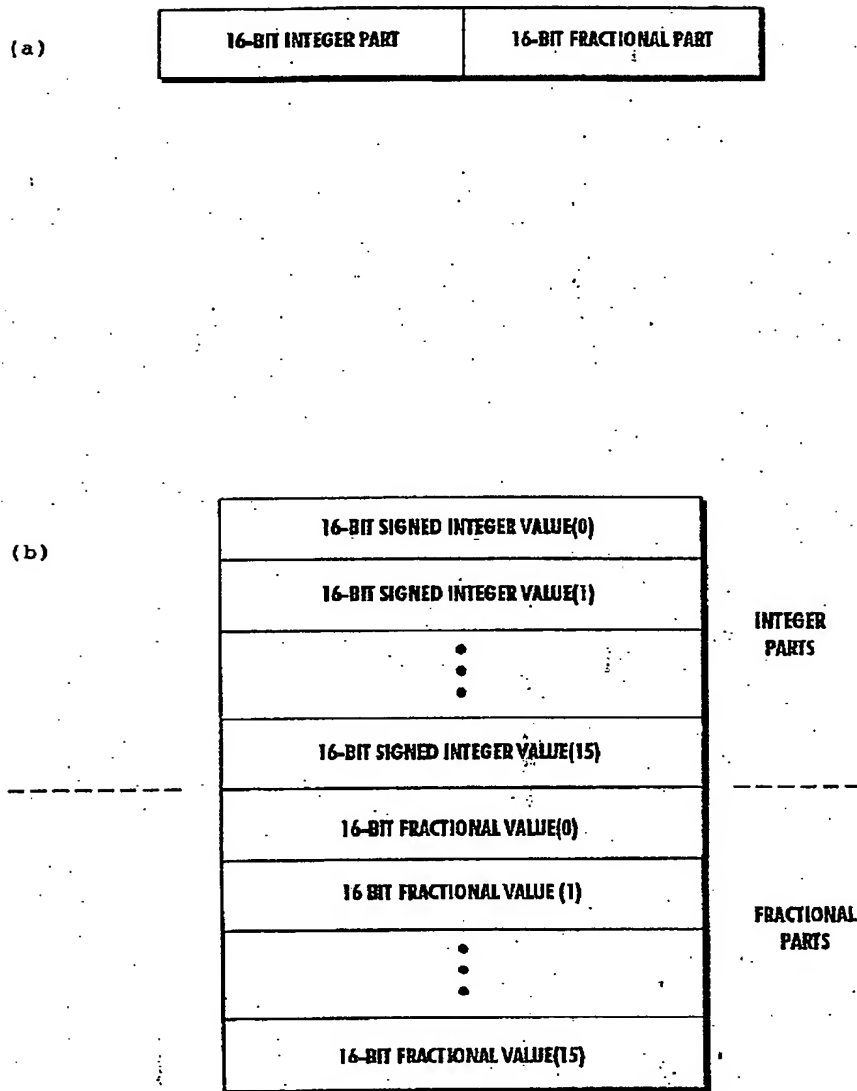


FIG. 22

(a)

VERTEX 0	408 (0)
VERTEX 1	408 (1)
VERTEX 2	408 (2)
⋮	
VERTEX 15	408 (15)

(b)

x.int	408 (1) (a)
y.int	408 (1) (b)
z.int	408 (1) (c)
w.int	408 (1) (d)
x.frac	408 (1) (e)
y.frac	408 (1) (f)
z.frac	408 (1) (g)
w.frac	408 (1) (h)
r	408 (1) (i)
g	408 (1) (j)
b	408 (1) (k)
a	408 (1) (l)
s	408 (1) (m)
t	408 (1) (n)
xsqr	408 (1) (o)
ysqr	408 (1) (p)
1/sqr.int	408 (1) (q)
1/sqr.frac	408 (1) (r)
t/w.int	408 (1) (s)
t/w.frac	408 (1) (t)
0c0c	408 (1) (u)
FLAGS (E.G. CLIP TEST RESULTS)	408 (1) (v)
00	408 (1) (w)

FIG. 23

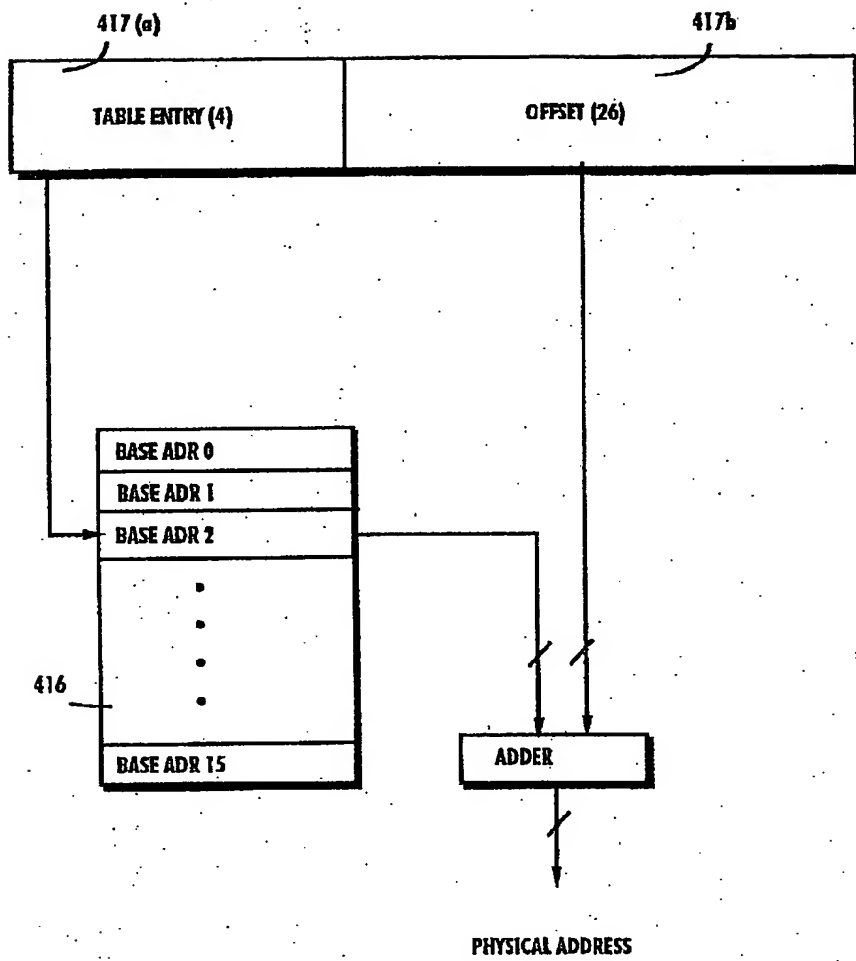


FIG. 24

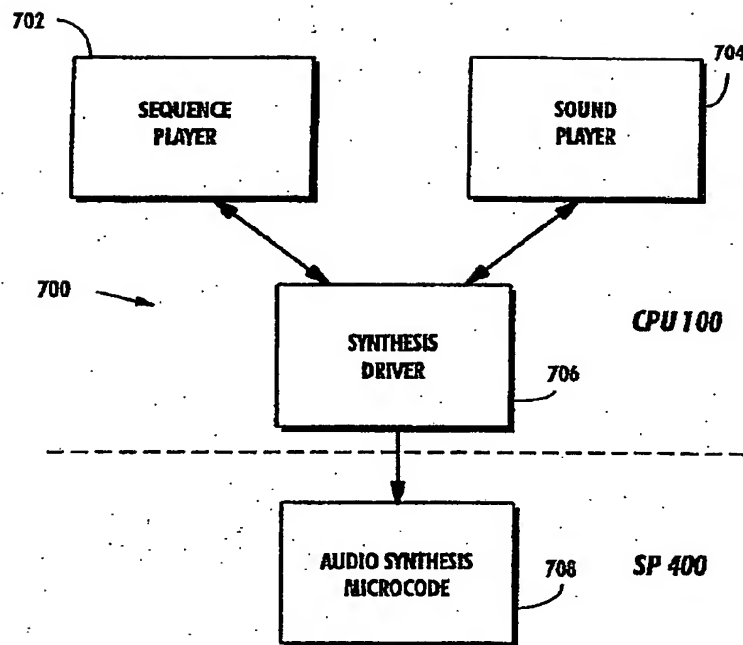


FIG. 25

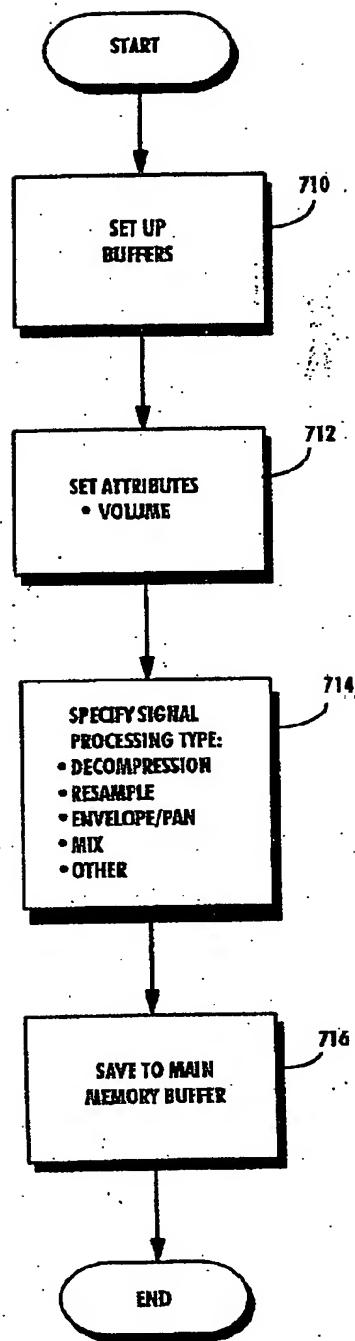


FIG. 26

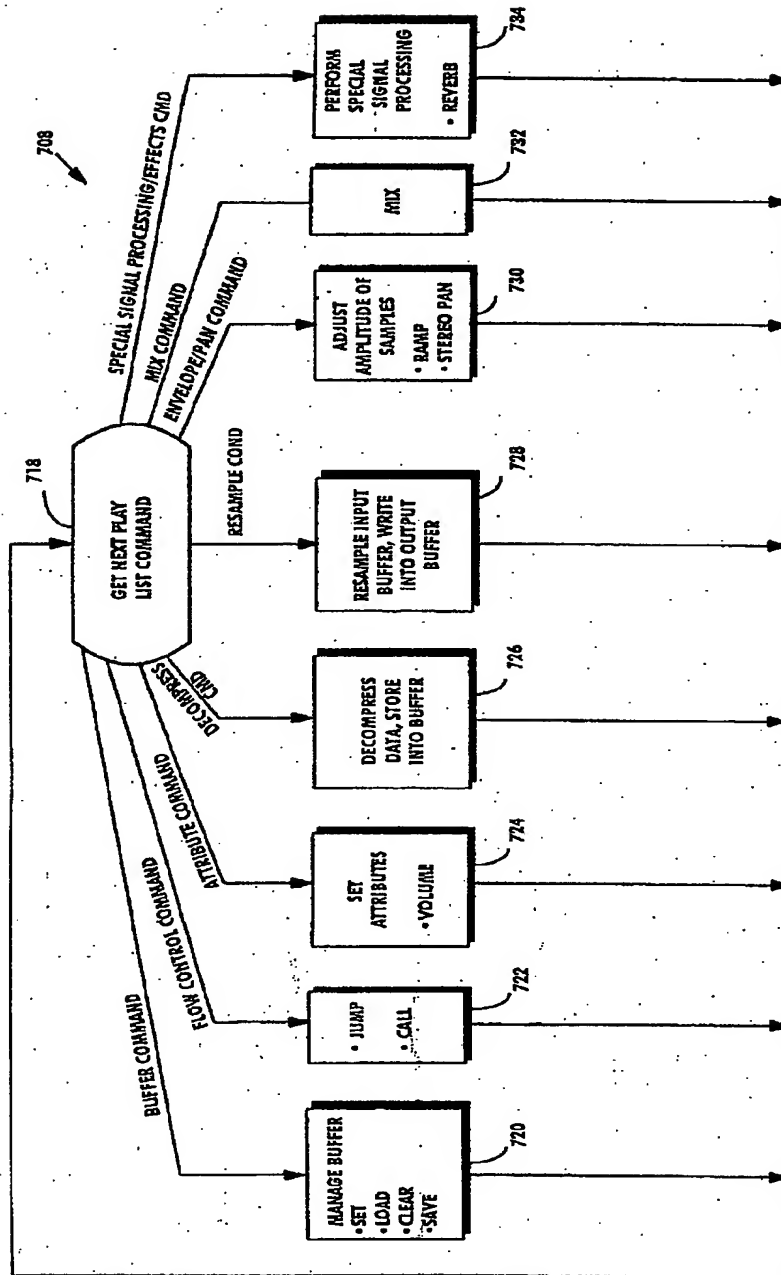
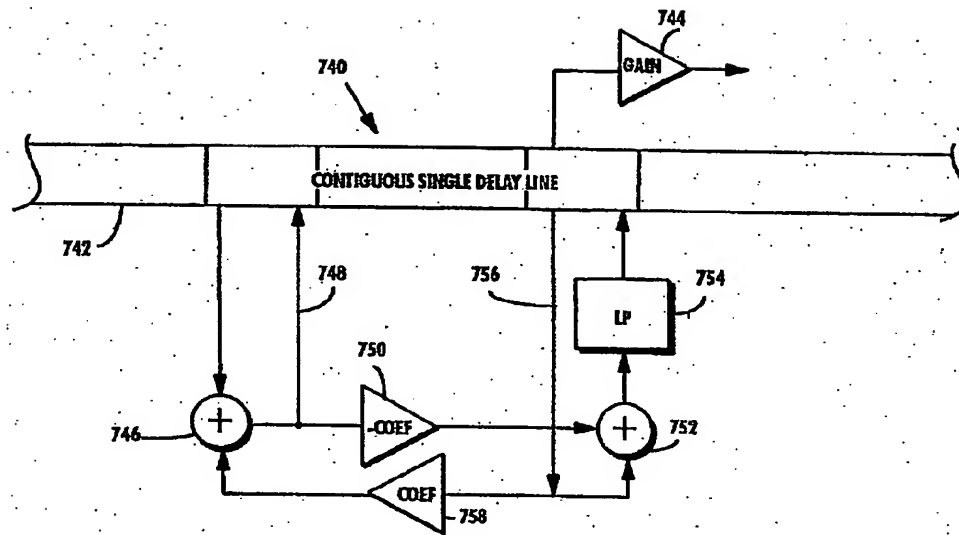


FIG. 27



**LOAD TILE/BLOCK,
LOAD TILT**

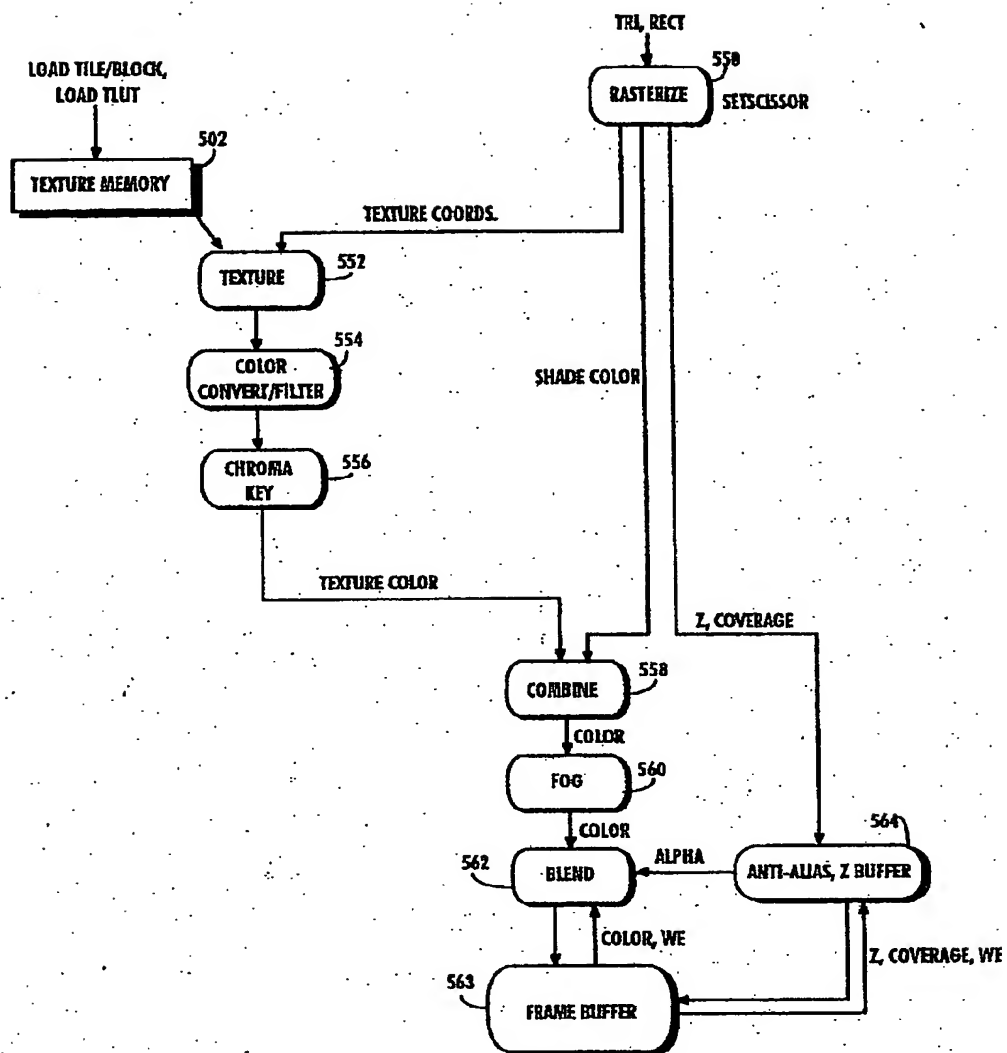


FIG. 29

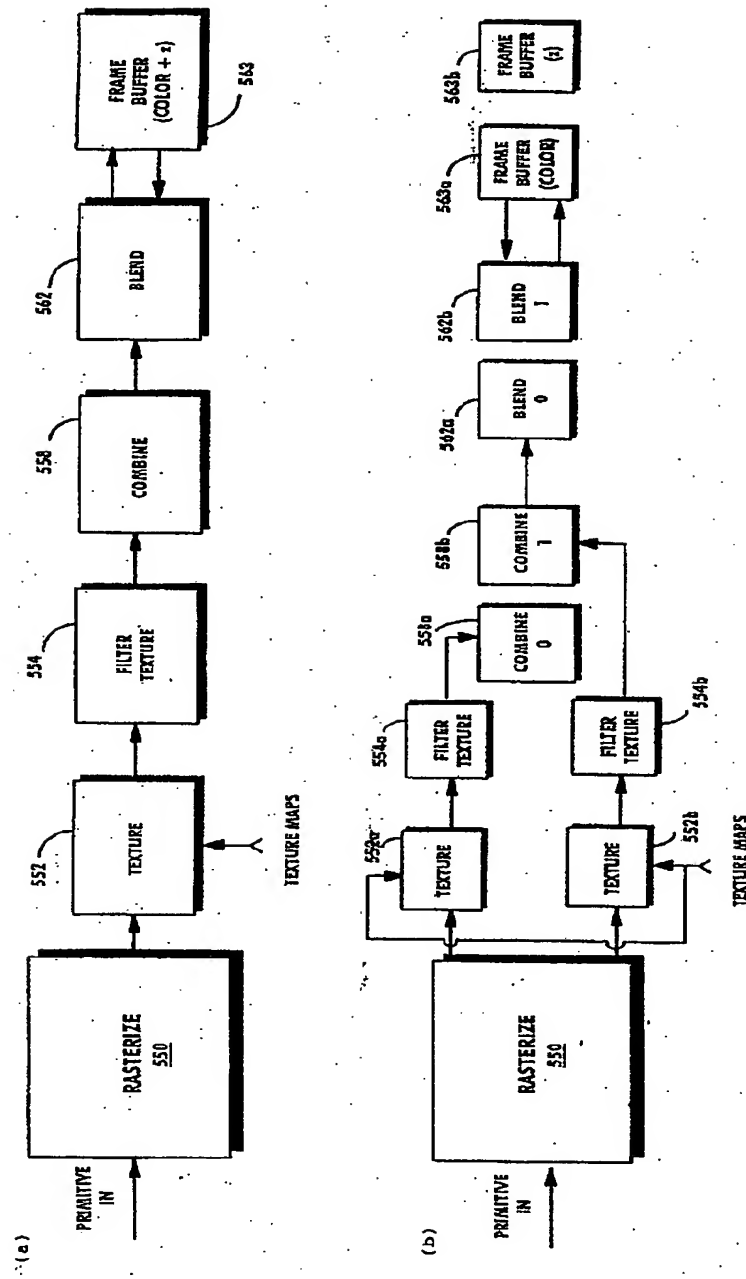


FIG. 30

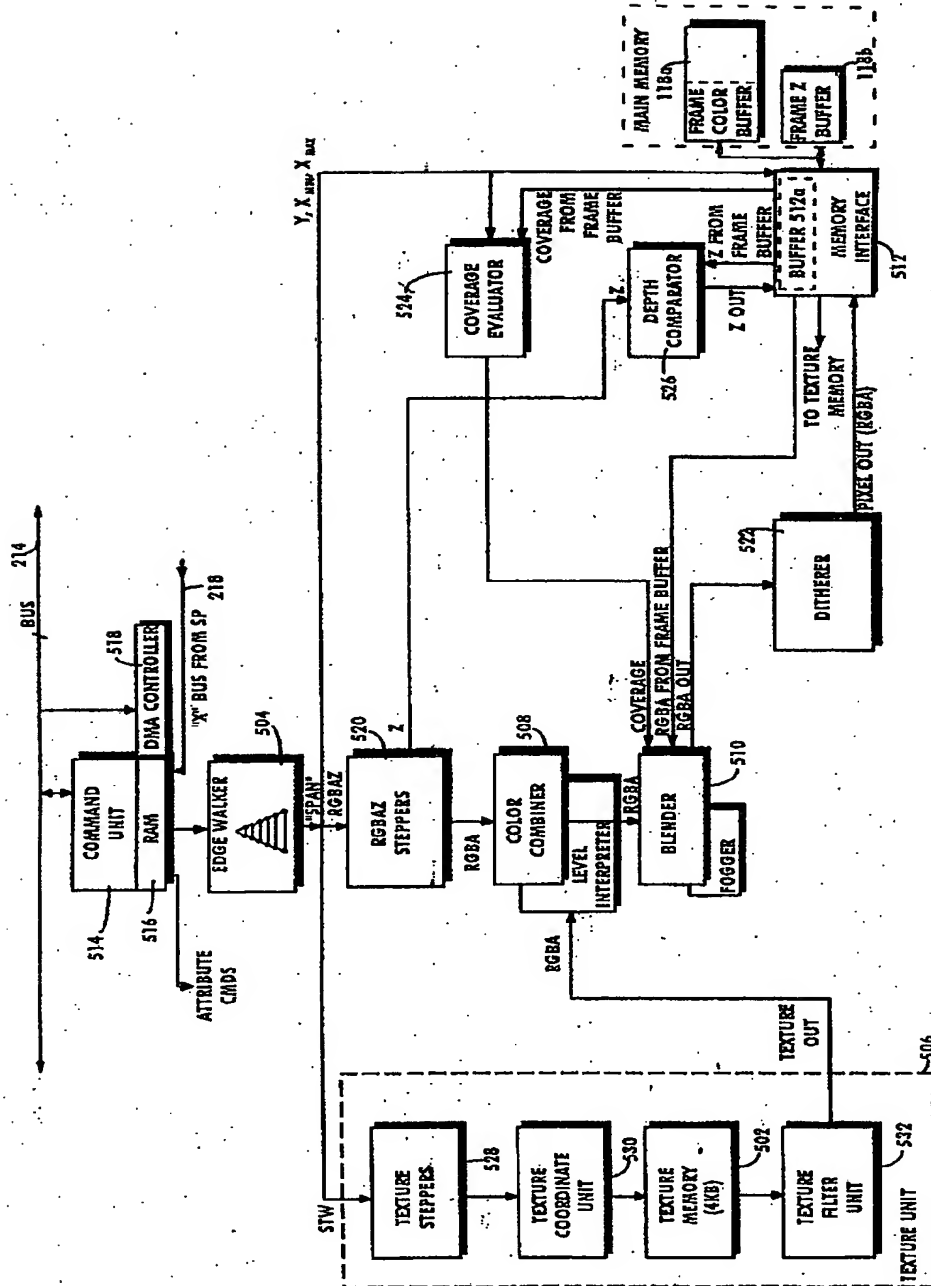


FIG. 31

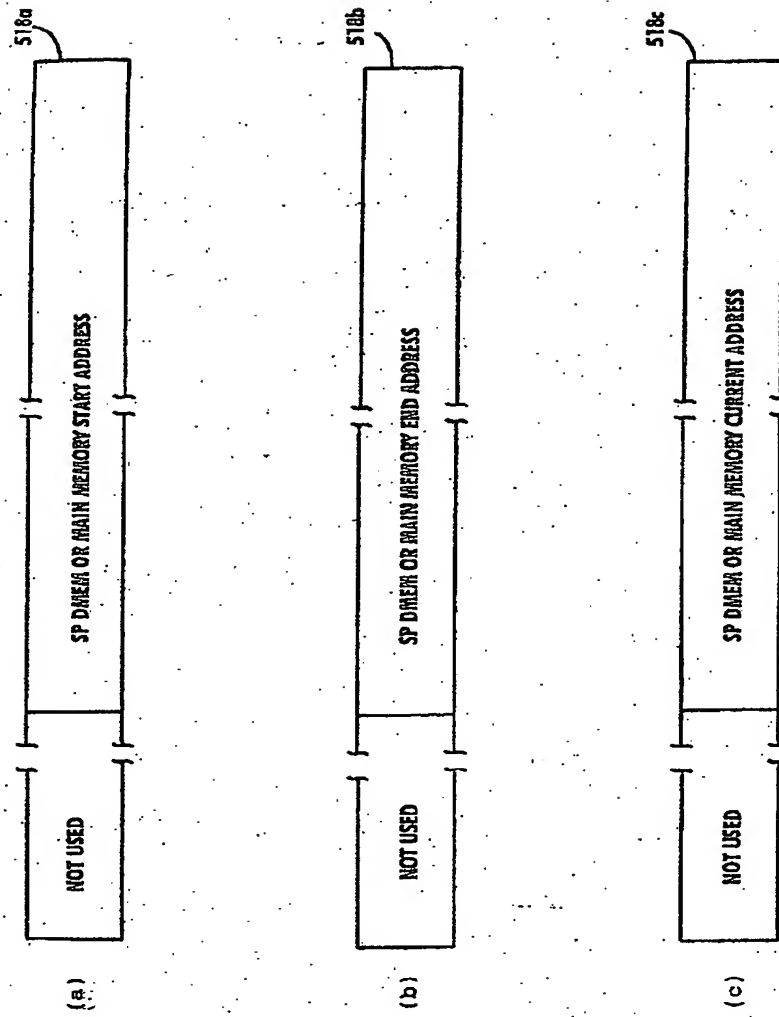


FIG. 32

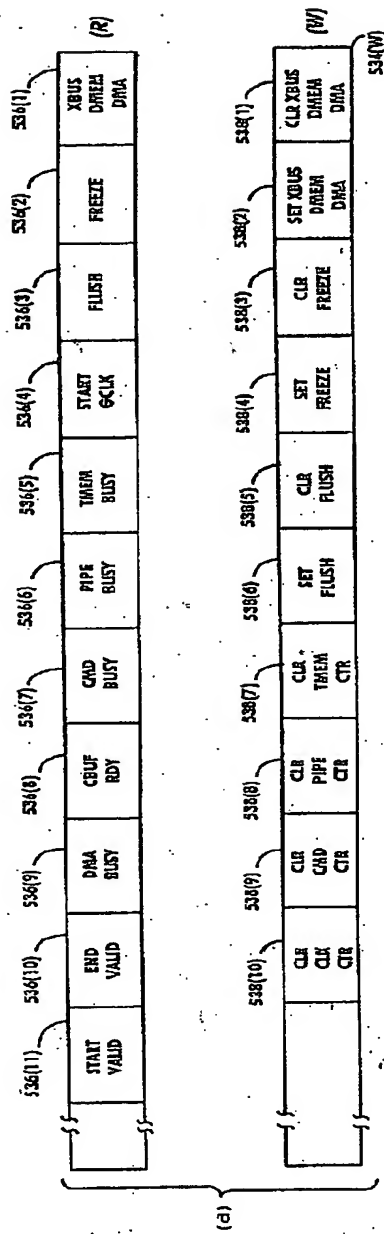


FIG. 33

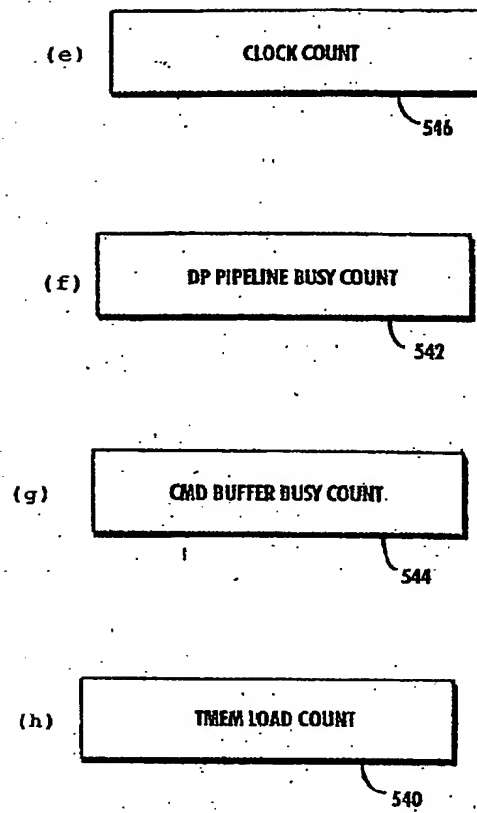


FIG. 34

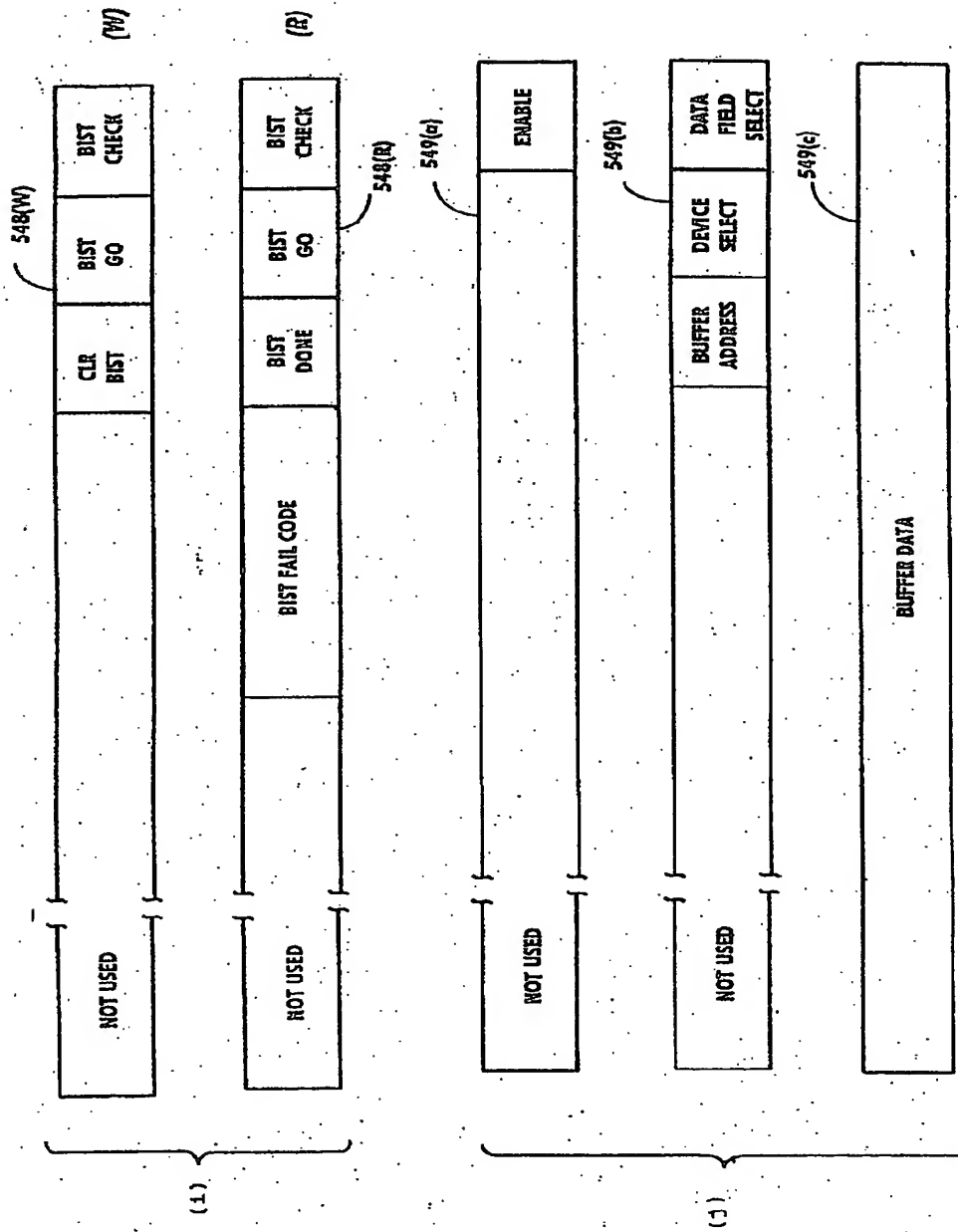


FIG. 35

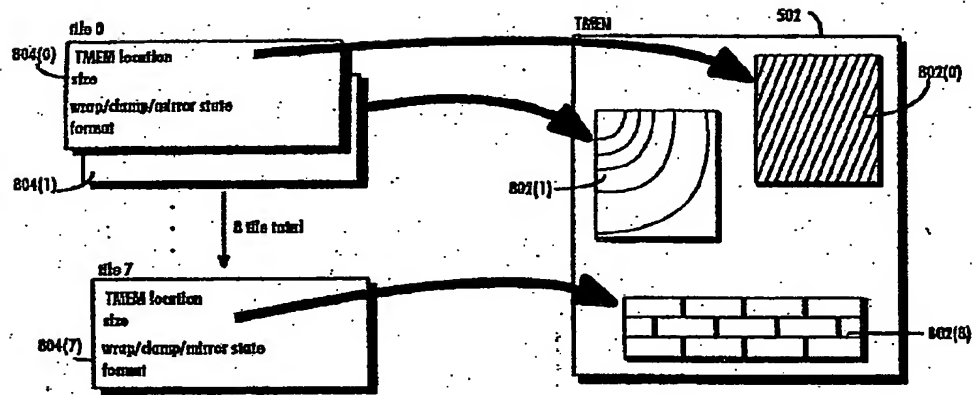


FIG. 36

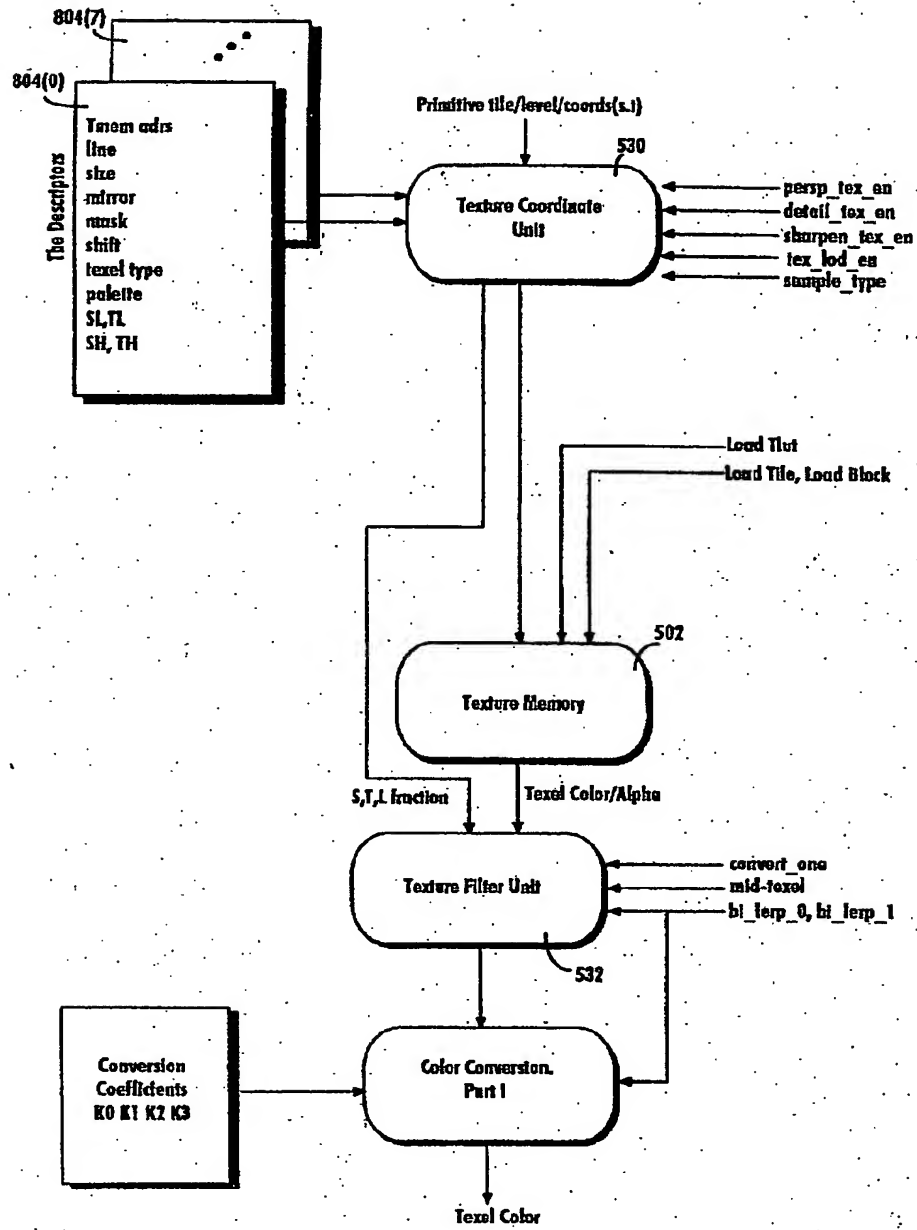


FIG. 37

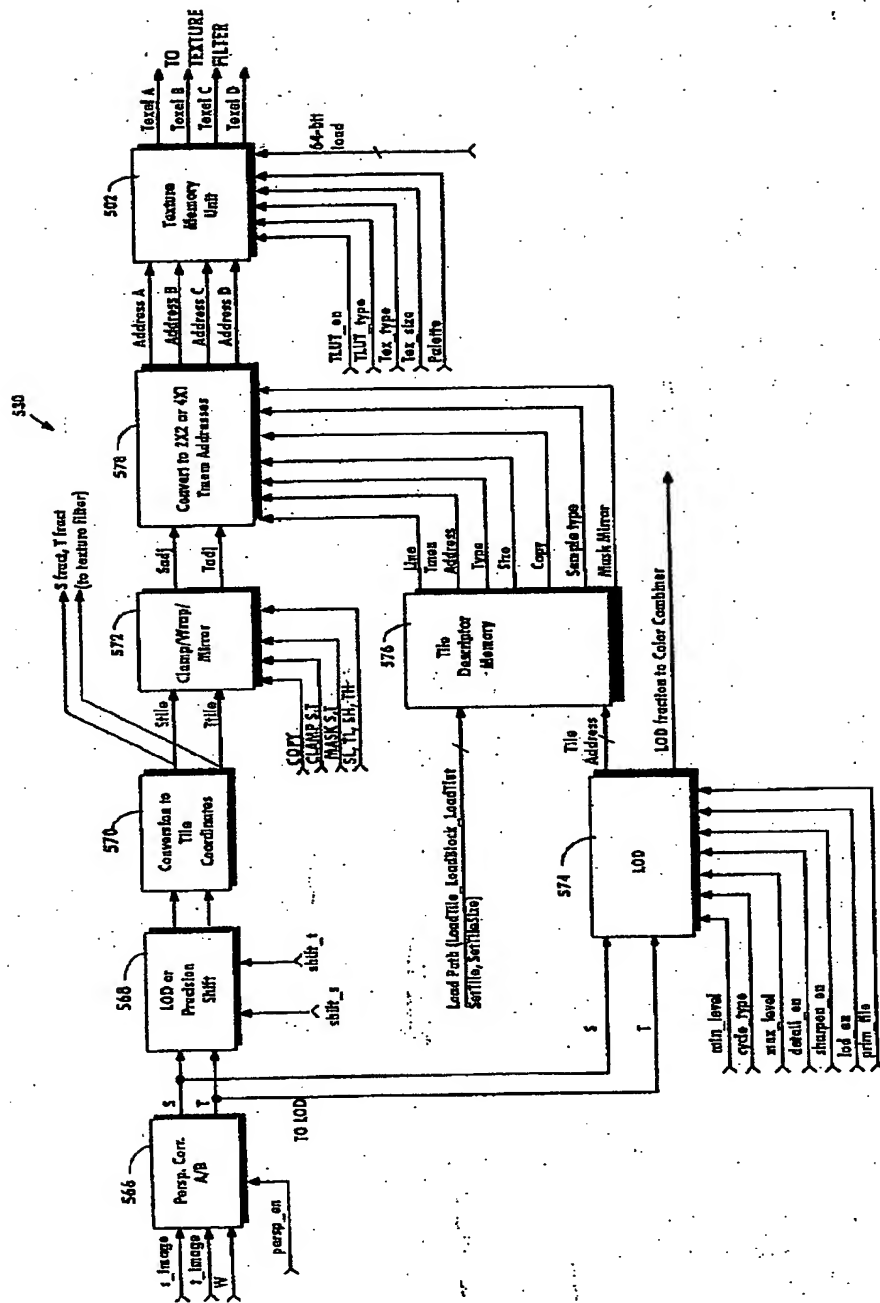


FIG. 38

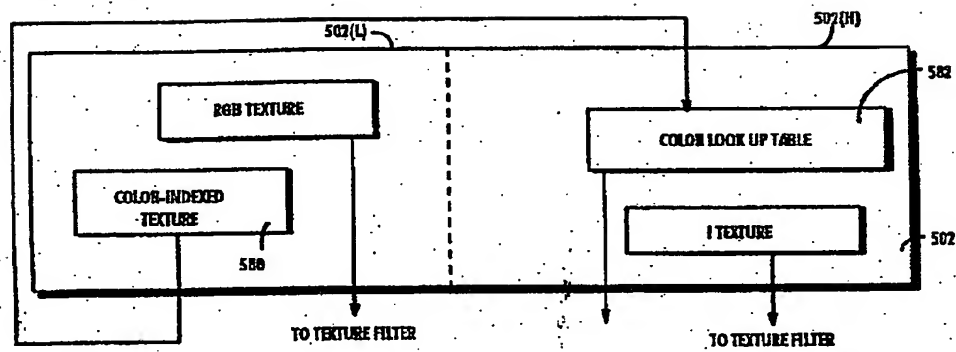


FIG. 39

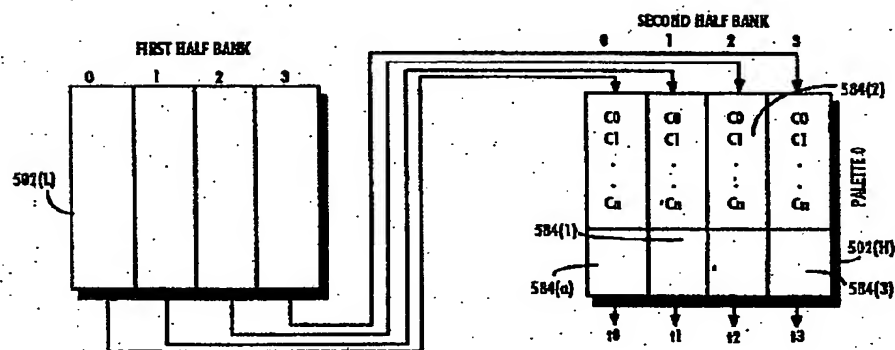
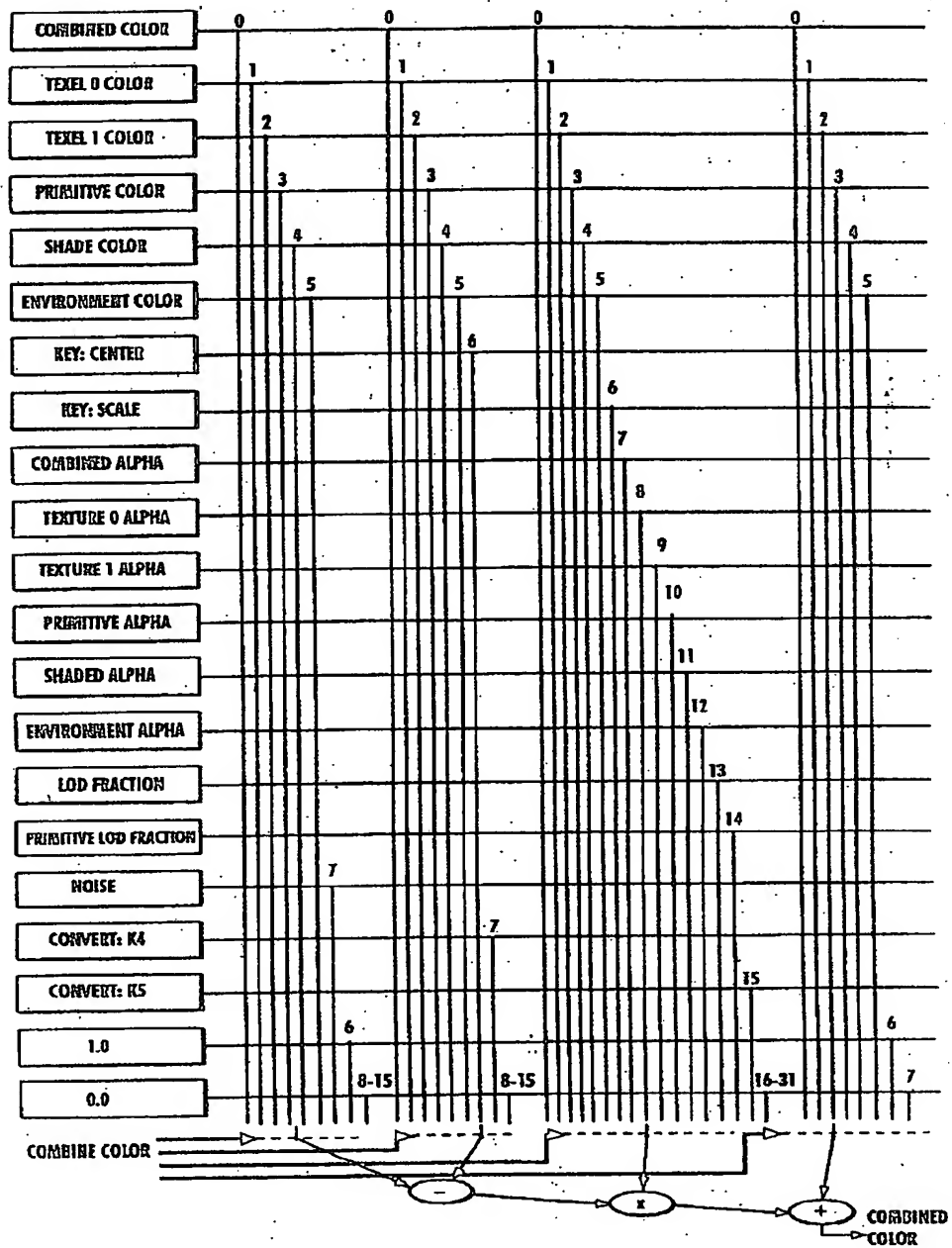


FIG. 40



The diagram illustrates the alpha blending process for a four-texture scene. It consists of two main parts: a timeline of alpha values and a corresponding alpha blending graph.

Alpha Values Timeline:

- COMBINED ALPHA:** 0, 0, 1, 0
- TEXTURE 0 ALPHA:** 1, 1, 1, 1
- TEXTURE 1 ALPHA:** 2, 2, 2, 2
- PRIMITIVE ALPHA:** 3, 3, 3, 3
- SHADED ALPHA:** 4, 4, 4, 4
- ENVIRONMENT ALPHA:** 5, 5, 5, 5
- LOD FRACTION:** 0, 6
- PRIMITIVE LOD FRACTION:** 6, 7
- 1.0:** 6, 7
- 0.0:** 7, 7

Alpha Blending Graph:

- The **COMBINE ALPHA** is shown as a dashed line with a step function.
- The **COMBINED ALPHA** is shown as a solid line with a smooth curve.
- The graph includes a subtraction node ($-$) and an addition node ($+$).
- The **COMBINED ALPHA** is calculated as the sum of the **COMBINE ALPHA** and the **PRIMITIVE ALPHA**.

FIG. 42

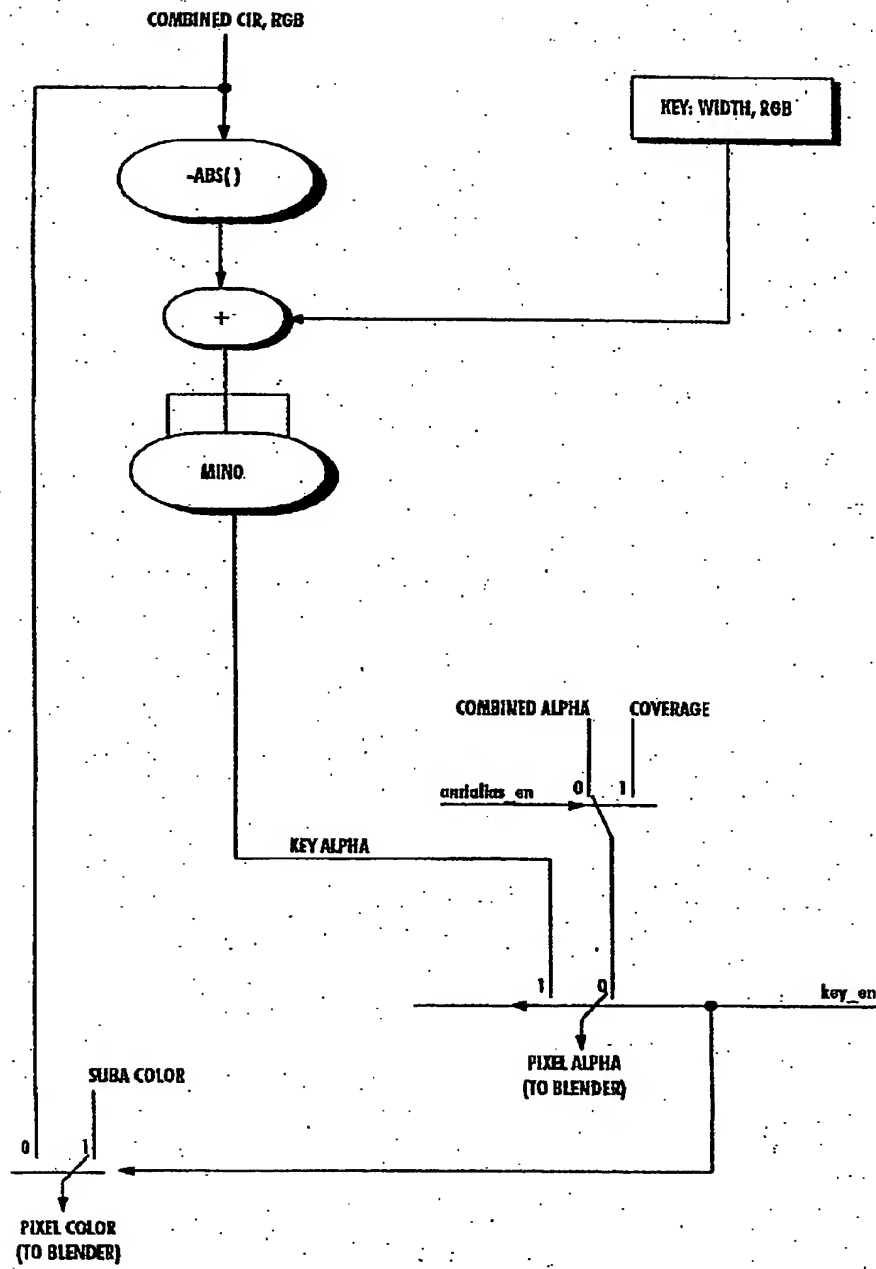


FIG. 48

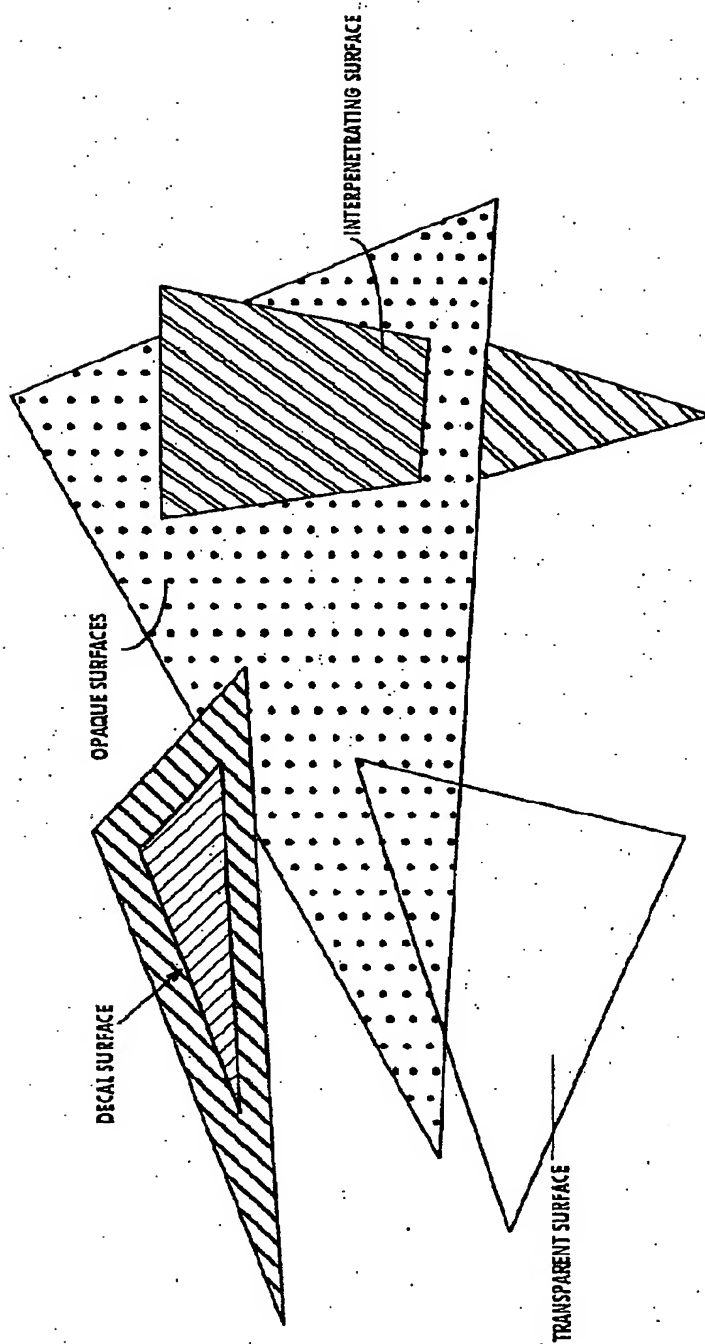


FIG. 44

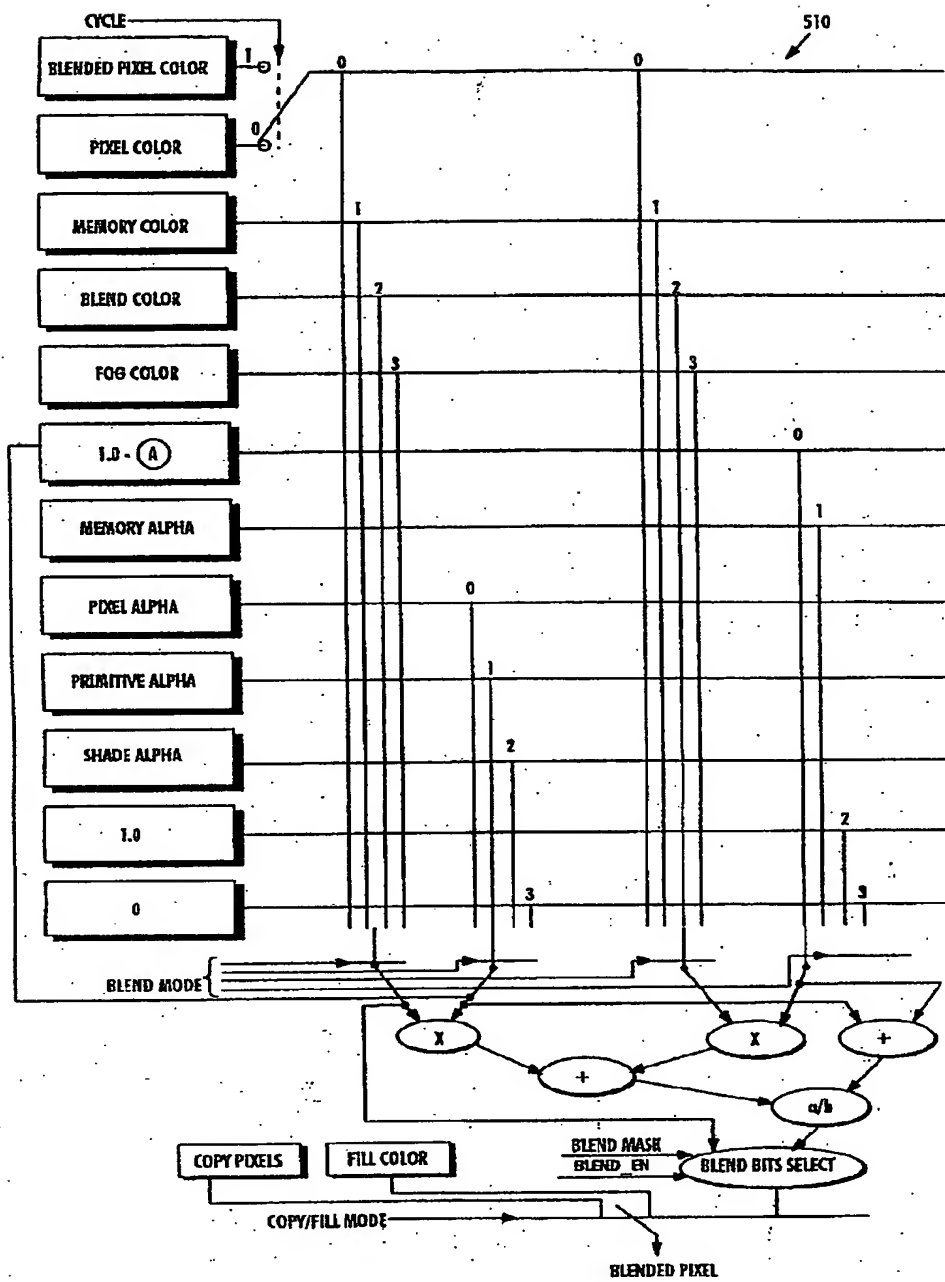


FIG. 45

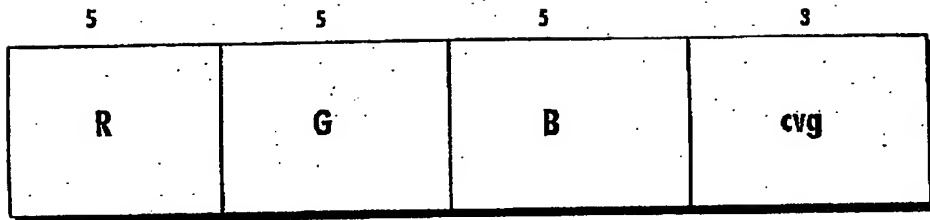
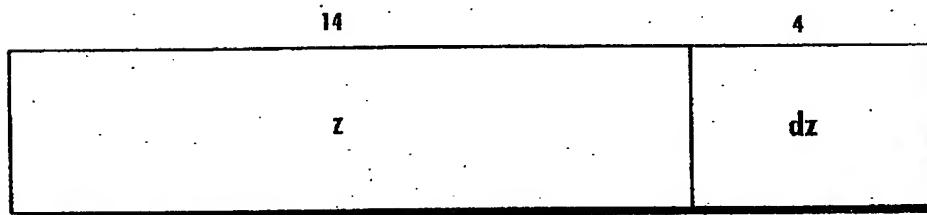


FIG. 46



The diagram illustrates the Z-Buffer logic. Key components and their connections include:

- Inputs:** PIXEL ALPHA, INDEX PIXEL, COPY PIXELS, COPY MODE, COLOR IMAGE TYPE, RANDOM DITHER ALPHA, BLEND COLOR ALPHA, PIXEL DELTA Z, PRIMITIVE DELTA Z, z_sel_sel, PIXEL Z, PRIMITIVE Z, MEMORY Z, near_z_cmp, far_z_cmp, a_compare_en, pf_sample, mask_15, coverage, z_update_en.
- Logic Blocks:**
 - COMPARE, >=**: Receives INDEX PIXEL and COLOR IMAGE TYPE. Its output goes to a multiplexer for z_sel_sel.
 - PRIMITIVE Z**: Receives PRIMITIVE Z and outputs to a multiplexer for PIXEL Z.
 - Arithmetic Blocks**: Includes adders (+) and subtractors (-). One adder calculates PIXEL DELTA Z + PRIMITIVE DELTA Z. Another calculates the difference between PIXEL Z and PRIMITIVE Z.
 - Shift Block**: A left shift (<<) block that takes the difference from the previous step and shifts it by delta_z_shift.
 - COMPARE**: Two comparison blocks. The first compares MEMORY Z with the result of the subtraction. The second compares MEMORY Z with the result of the addition.
 - Multiplexers**: Several 2-to-1 multiplexers are used to select between different data paths based on control signals like a_compare_en, pf_sample, mask_15, and coverage.
 - AND/OR Gates**: Used to combine signals like z_compare_en, behind, infront, and force_blend to produce the final Z-Buffer Write Enable.
- Outputs:** COLOR WRITE ENABLE, Z-Buffer Write Enable.

FIG. 48

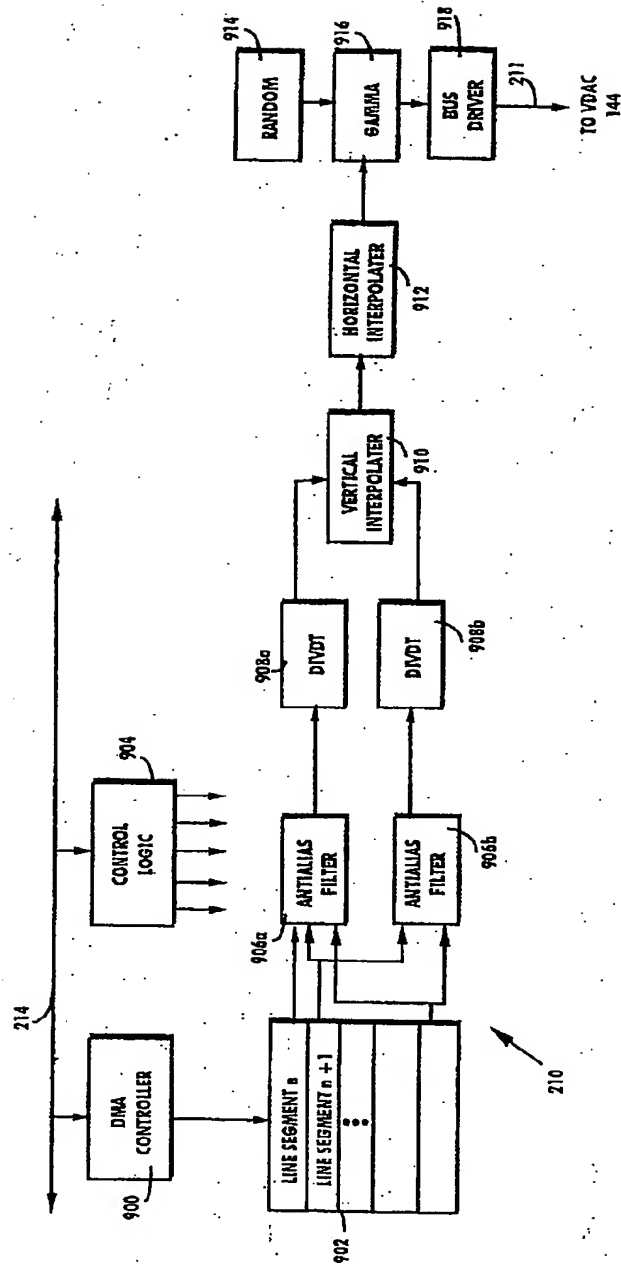


FIG. 49

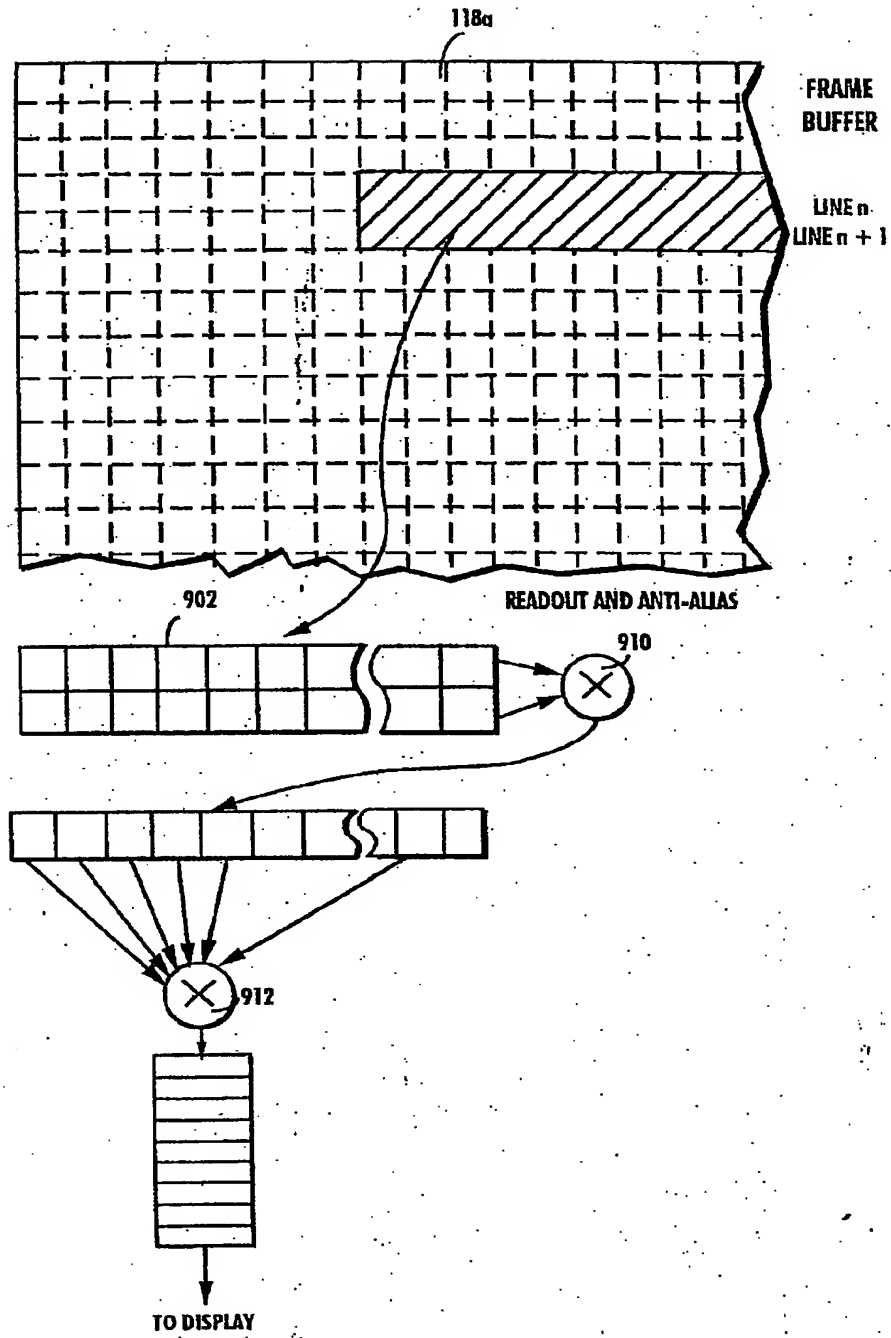


FIG. 50

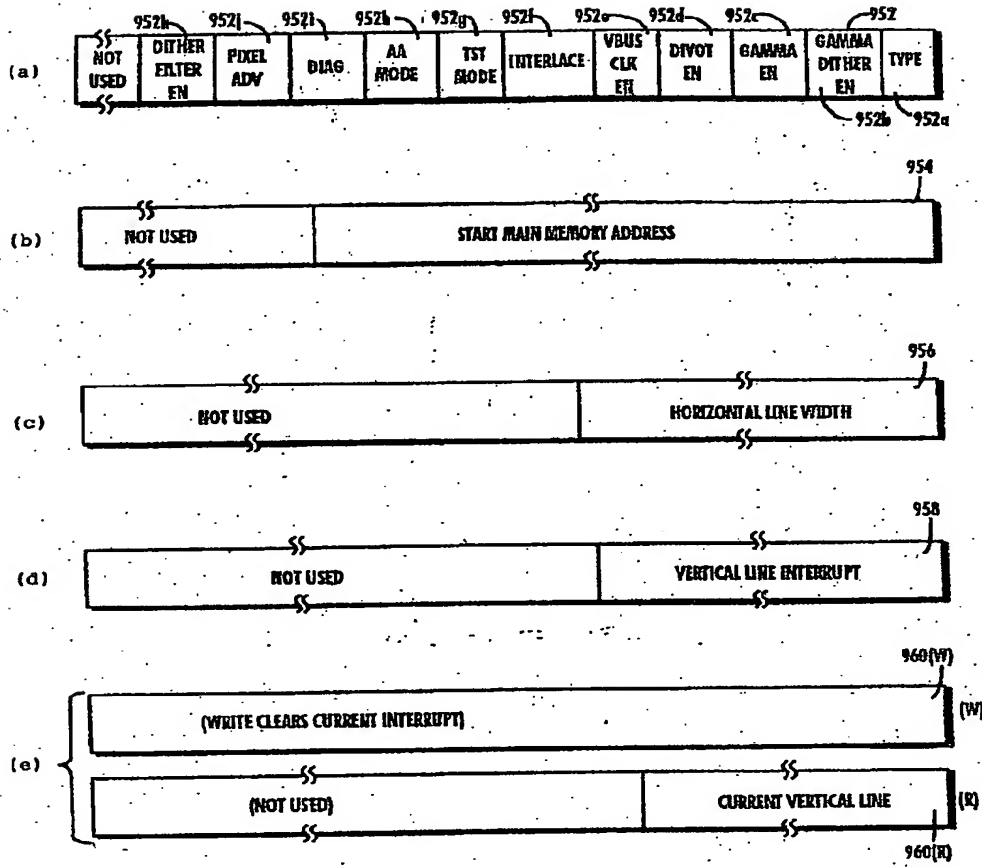


FIG. 51

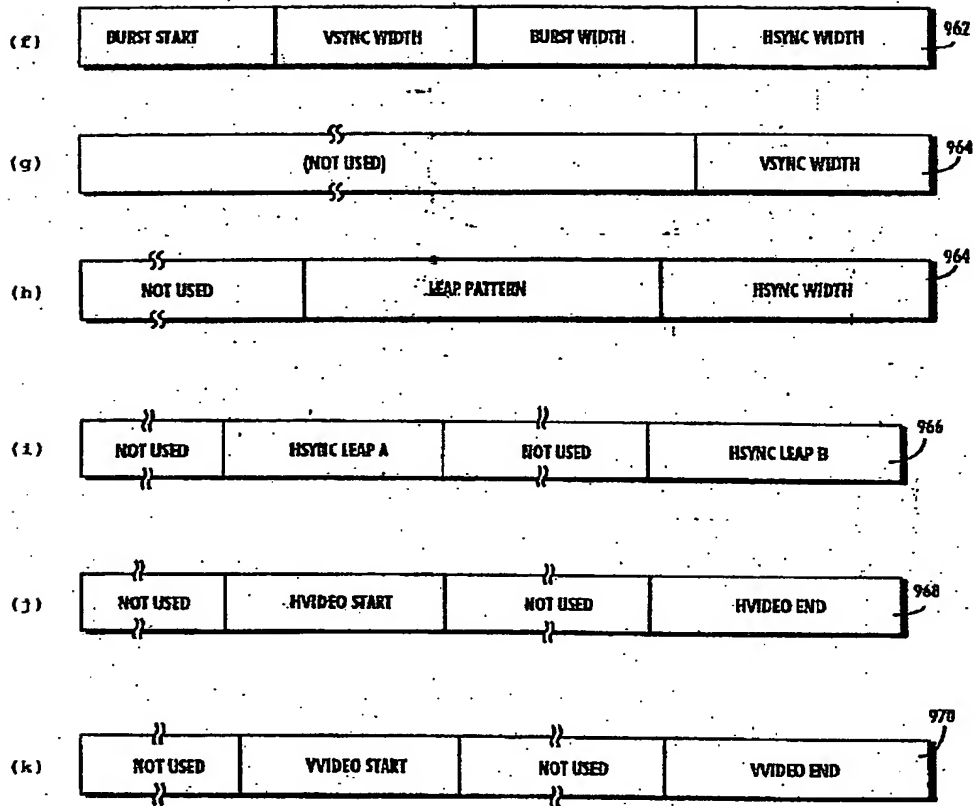


FIG. 52

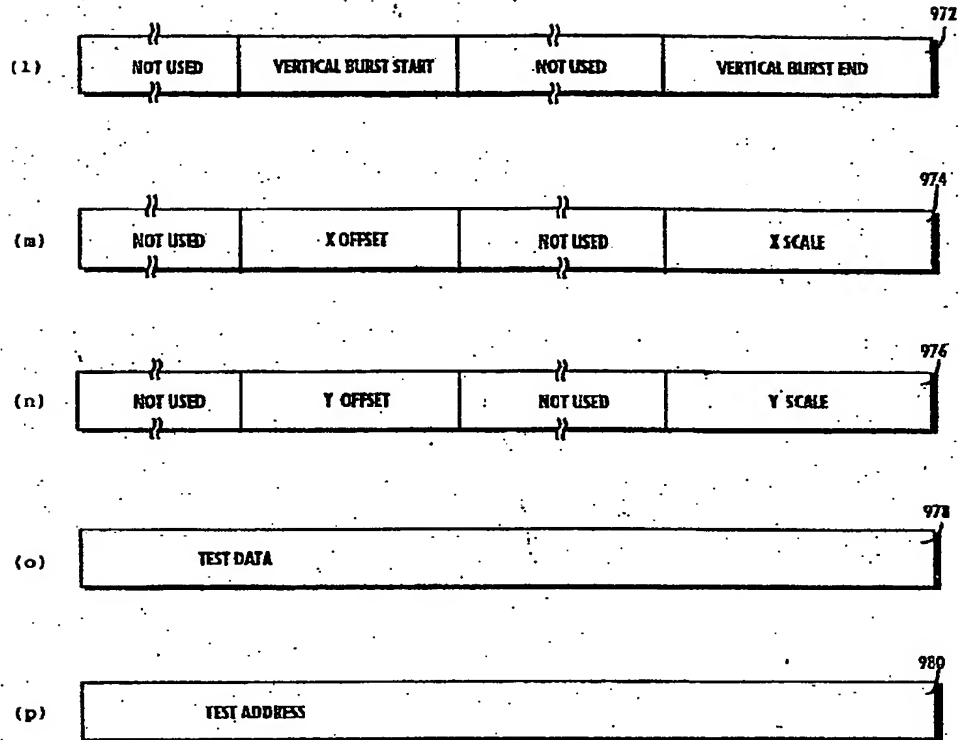


FIG. 53

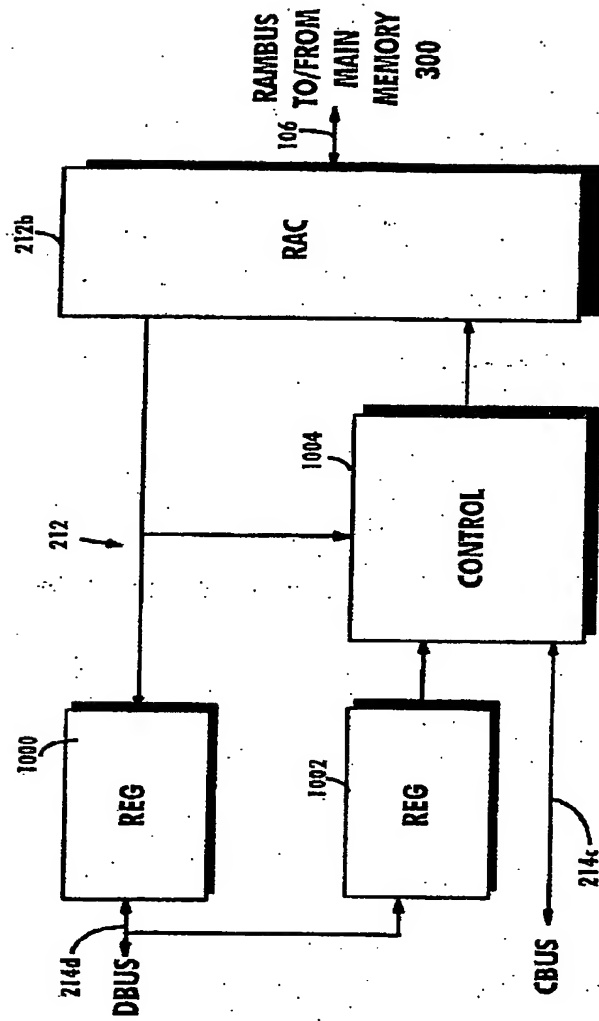


FIG. 54

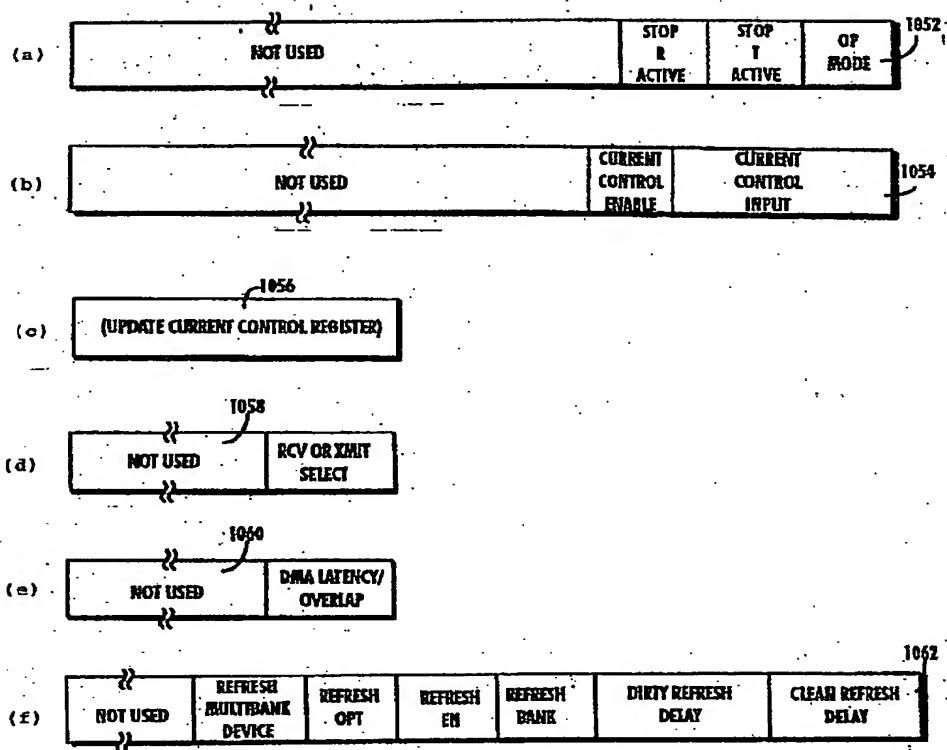


FIG. 55

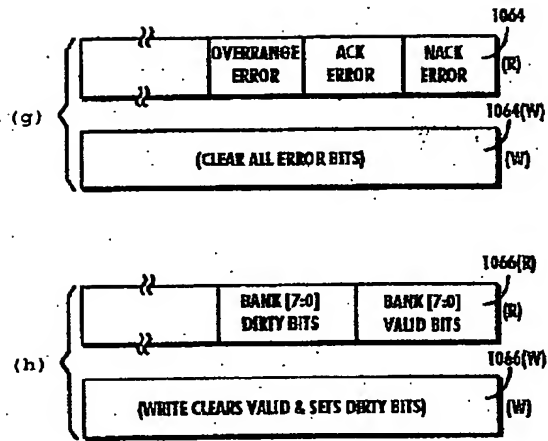


FIG. 56

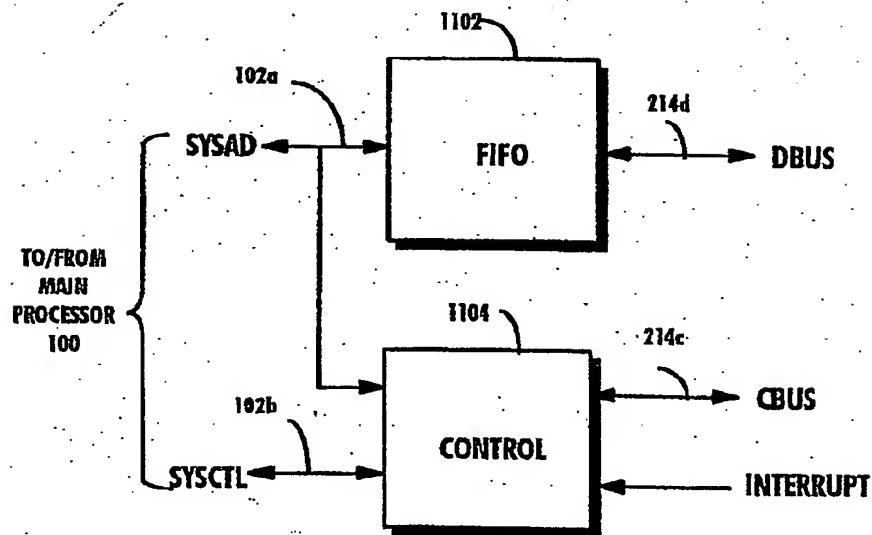


FIG. 57

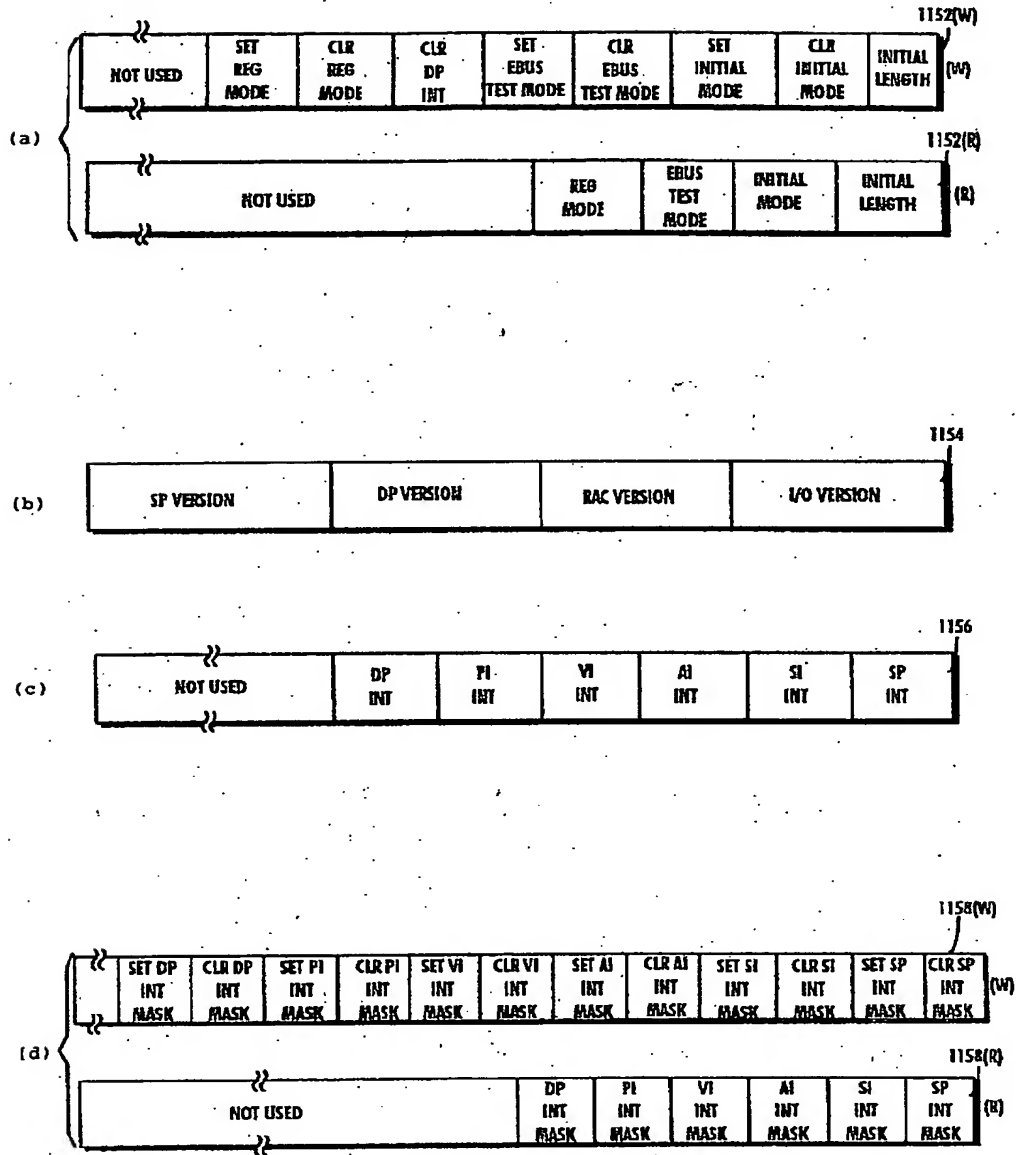


FIG. 58

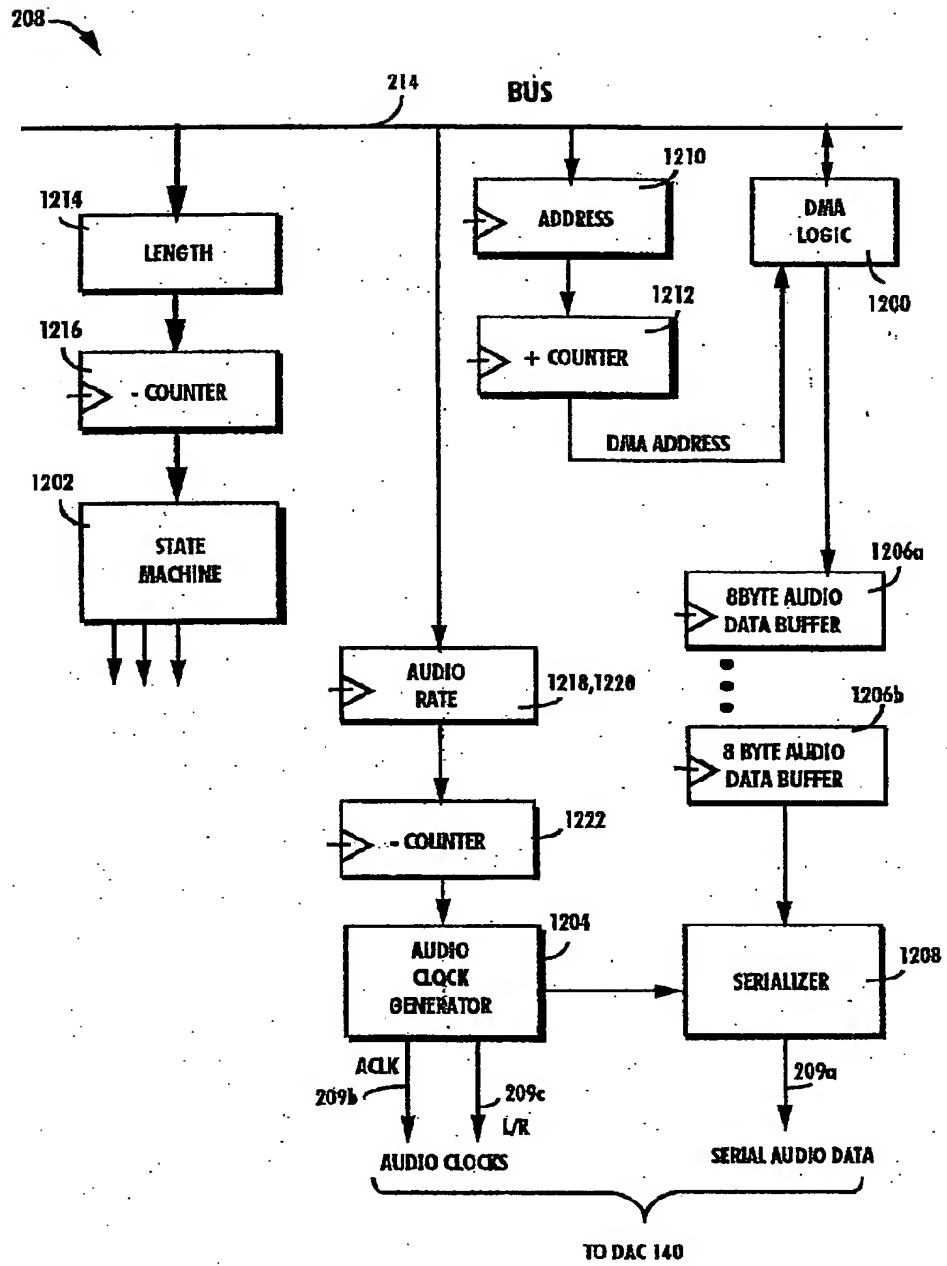


FIG. 59

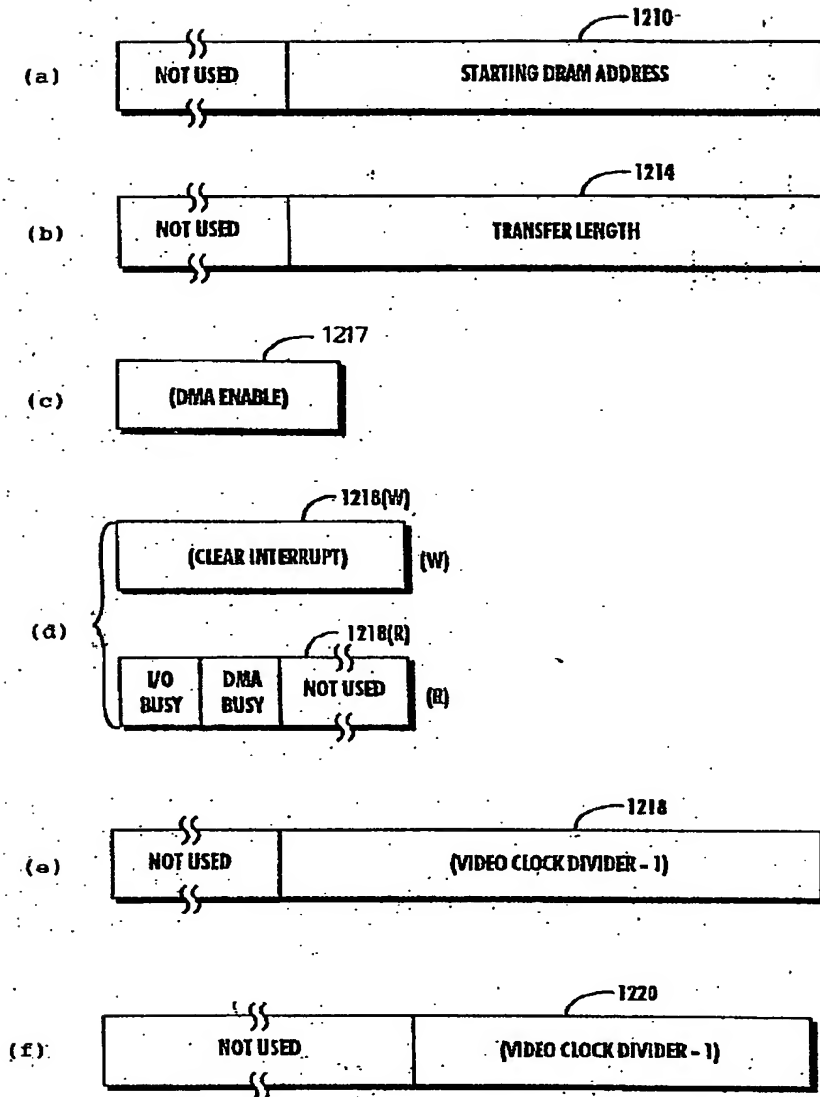


FIG. 60

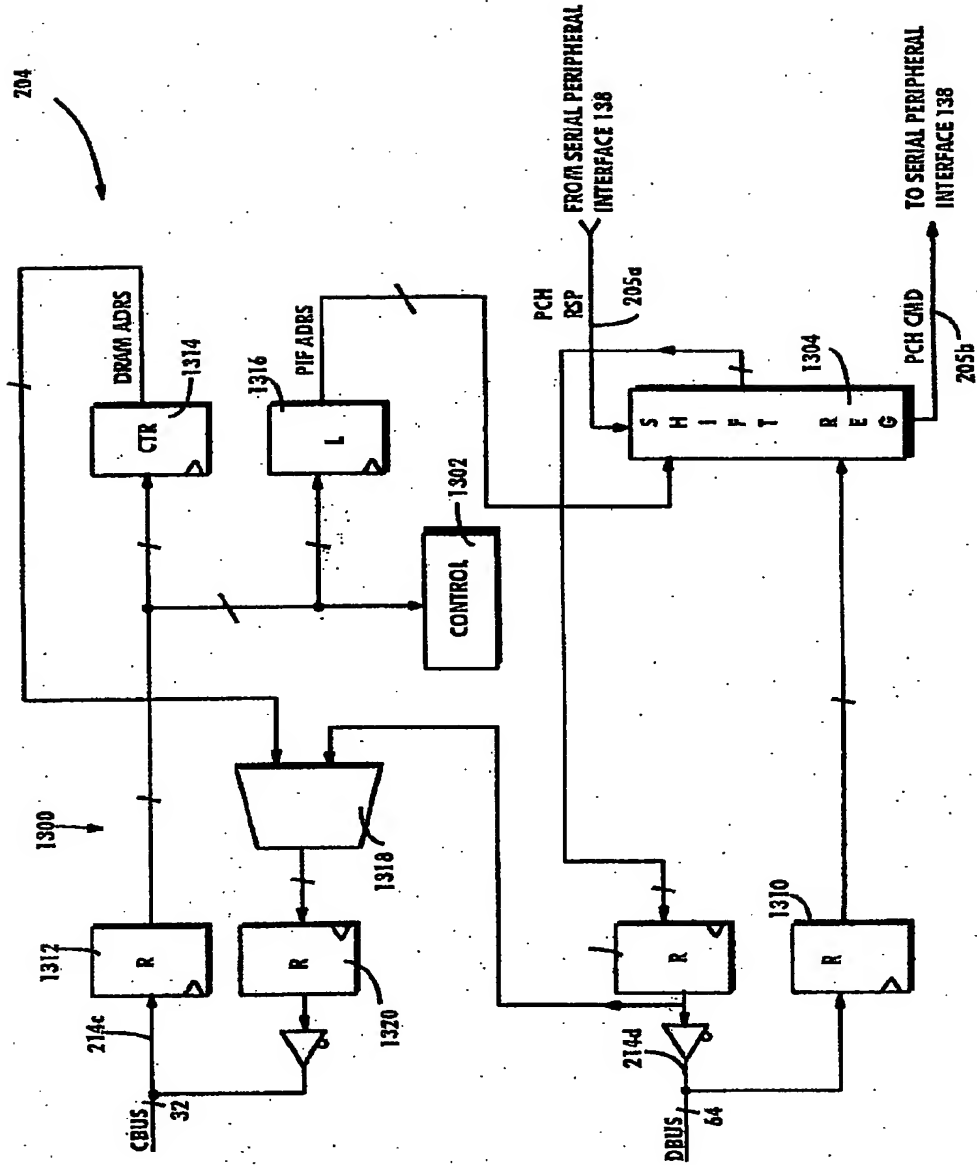


FIG. 61

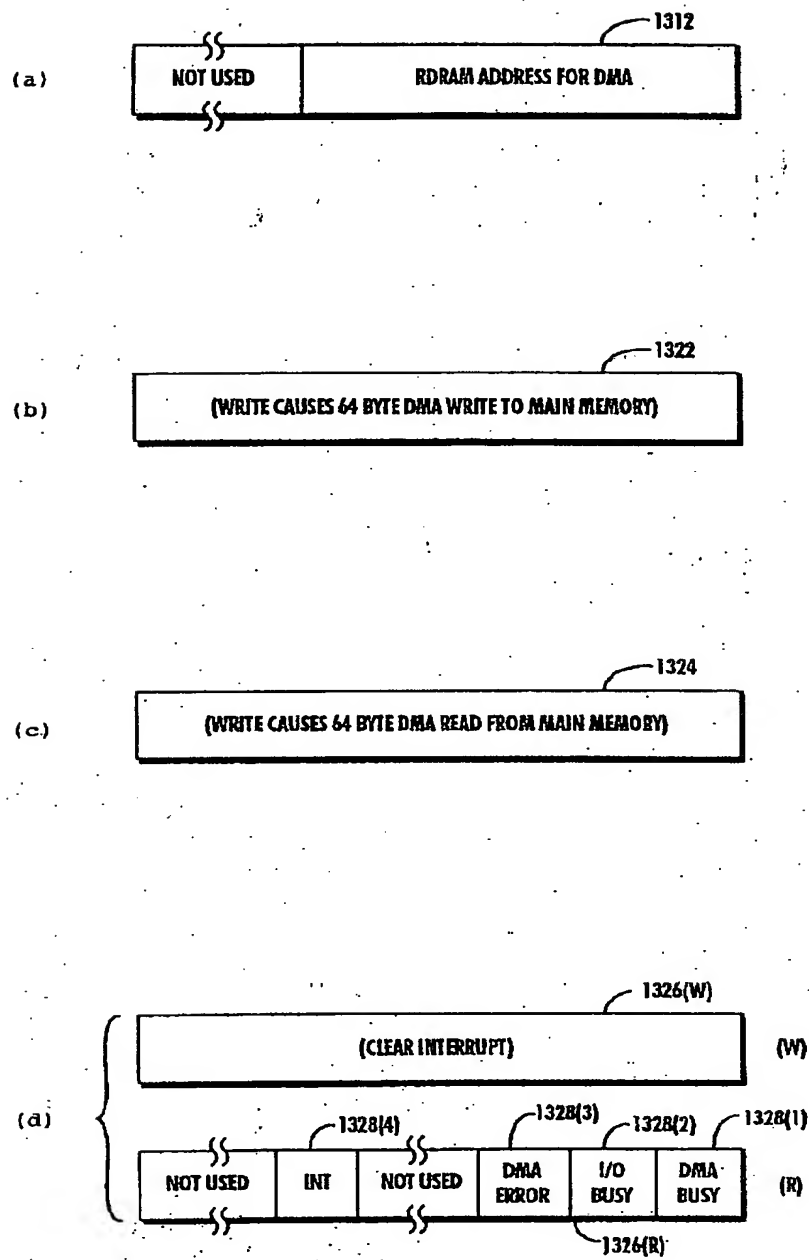


FIG. 63

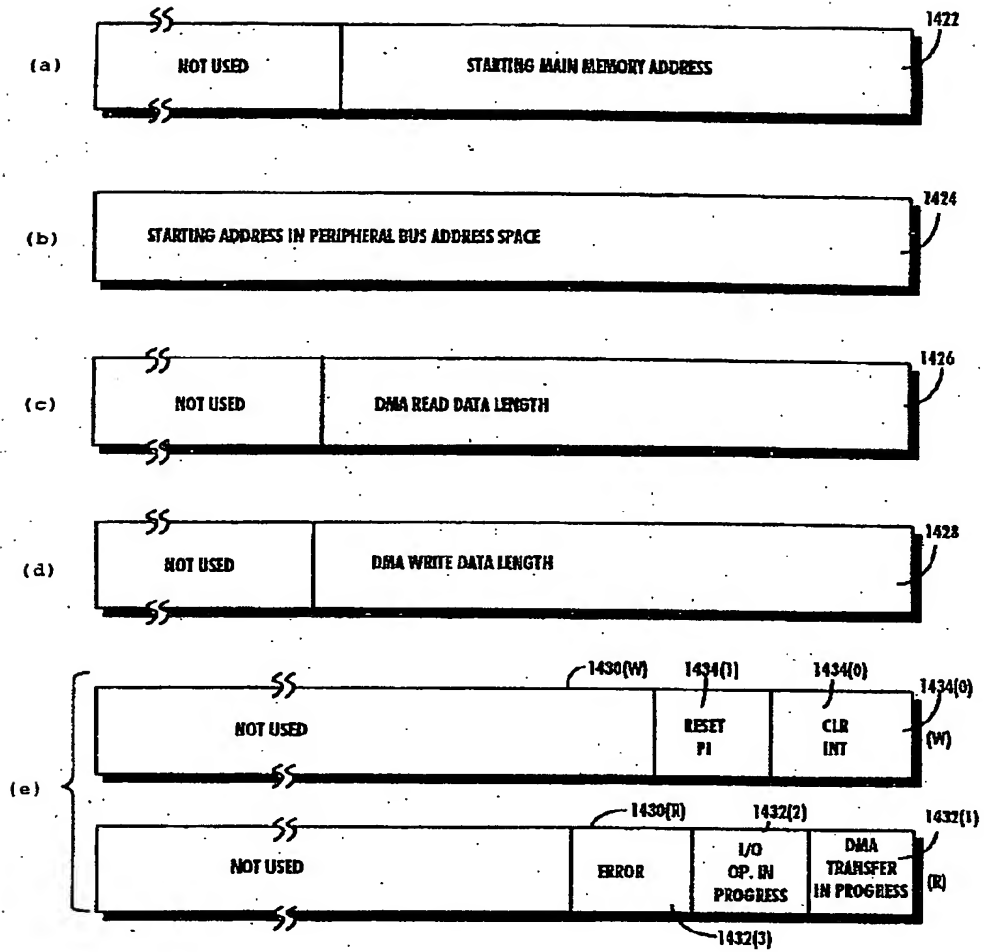


FIG. 64

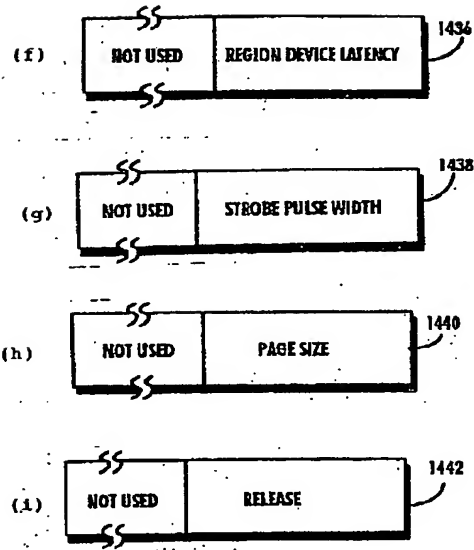
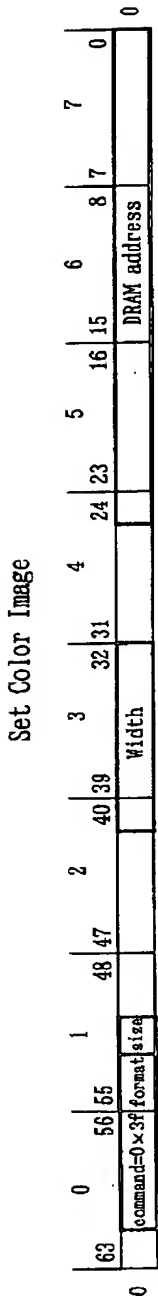


FIG. 65



Set_Color_Image Command Format

Field	Word	Bits	Description
Command	0	61-56	Command identifier
format	0	55-53	Image data format, 0=rgba, 1=yuv, 2=Color Indx, 3=1A, 4=1
size	0	52-51	Size of pixel/textel color element, 0=4b, 1=8b, 2=16b, 3=32b
Width	0	41-32	Width of image in pixels, image width=width+1
DRAM adrs	0	25-0	Base address(top left corner) of image in DRAM, in bytes

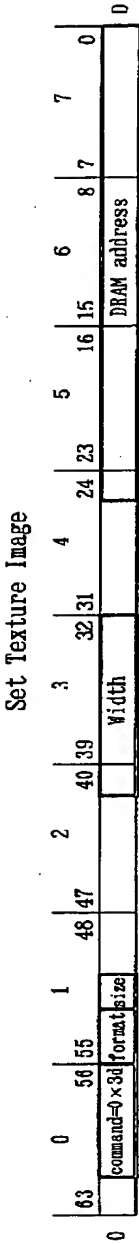
Legal Color Image Types/Sizes

Type	8b	16b	32b
RGBA		✓	✓
YUV			
Clr Indx	✓		

Set_Color_Image Usage Notes:

Read/Modify/Write of 32b color image with depth buffer must be done in two cycle mode.

FIG. 66



Set_Texture_Image Command Format

Field	Word	Bits	Description
command	0	81-56	Command identifier
format	0	55-53	Image data format, 0=rgba, 1=yuv, 2=Color Indx, 3=IA, 4=i
size	0	52-51	Size of pixel/texture color element, 0=4b, 1=8b, 2=16b, 3=32b
Width	0	41-32	Width of image in pixels, image width=width+1
DRAM adrs	0	25-0	Base address(top left corner) of image in DRAM, in bytes

Legal Texture Image Types/Sizes			
Type	4b	8b	16b 32b
RGBA			✓
YUV			✓
Clr Indx	✓	✓	
IA	✓	✓	✓
I	✓	✓	

FIG. 67

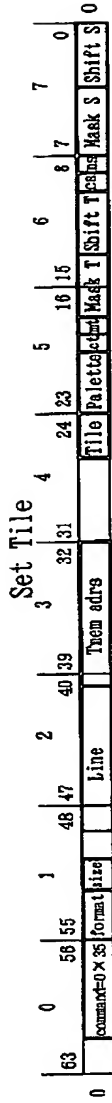
Set Z Image

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
command=0×3e																						0
DRAM address																						0

Set_Mask_Image Command Format

Field	Word	Bits	Description
command	0	61-56	Command identifier
DRAM adrs	0	25-0	Base address(top left corner) of image in DRAM, in bytes

FIG. 68



Set Tile Command Format

Field	Word	Bits	Description
command	0	61-56	Command Identifier
format	0	55-53	Image data format, 0=rgba, 1=yuv, 2=Color Indx, 3=IA, 4=I
size	0	52-51	Size of pixel/texture color element, 0=4b, 1=8b, 2=16b, 3=32b, 4=Other
Line	0	49-41	Size of tile line in 64b words, max of 4KB
Tile Adrs	0	40-32	Starting Tile address for this tile in words(64b), 4KB range
tile	0	28-24	Tile descriptor Index
Palette	0	23-20	Palette number for 4b Color Indexed texels. This number is used as the MS 4b of an 8b index.
ct	0	19	clamp enable for T direction
nt	0	18	mirror enable for T direction
Mask T	0	17-14	Mask for wrapping/mirroring in T direction. If this field is zero then clamp, otherwise pass(mask)LSBs of T address.
Shift T	0	13-10	Level of Detail shift for T addresses
cs	0	9	clamp enable bit for S direction
ms	0	8	mirror enable bit for S direction
Mask S	0	7-4	Mask for wrapping/mirroring in S direction. If this field is zero then clamp, otherwise pass(mask)LSBs of S address.
Shift S	0	3-0	Level of Detail shift for S address

Set Tile Usage Notes:

For YUV textures, Tile Line(number of Tile words per Tile line) is Line=(Width+7)>>3(8b texels) because although the image texels are 16bit, the Tile Y texels are 8 bit and the Tile UV texels are 16 bit at 1/2 the width in S.

YUV mask and mask/mirror are undefined

No mirroring for 32b RGBA Images.

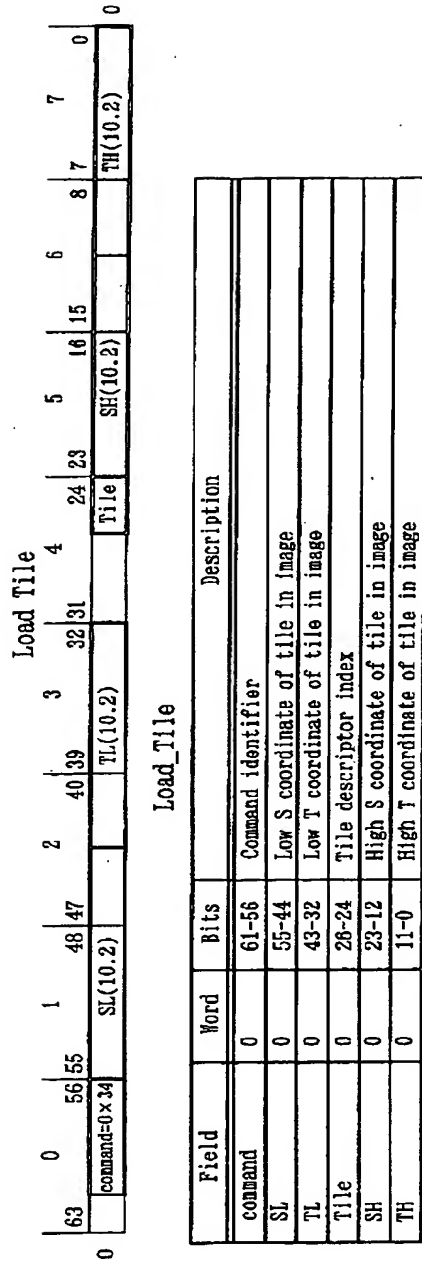
Wrap on all but YUV Images.

YUV texture Images are stored as interleaved 8bit YUV as 16bits per texel. Software must specify texture coordinates and tile coordinates and sizes which are even numbers in S(16b=0), so that a pair UV values are available. Software must specify a Tile Base address in the low half of Tile, and load a tile which fits in the low half (where 2048<Base+7) in 8b Y's. Software can't have CI TUV's and YUV textures coexisting in Tile.

32b RGBA, YUV Set Tile type should indicate 16b texels.

Mask==j means pass bit 0 so minimum mask width is 2 texels

FIG. 69



Load Tile Usage Notes:

4b textures should be loaded as bytes(they must be byte aligned) using the Set_Image Texture type field when loading more than 4k texels. This means the Load_Block parameters will have 8b texel units. The Set_Tile type field can be used to set the proper 4b type.

The Set_Tile_Sz can be used to set the proper SL,SH,TL,TH values after the Load_Block.

In general, textures can be loaded as one type and used as another type by proper manipulation of the Set_Image.Texture and Set_Tile format size fields.

Normally, during a load, the tile texel size and image pixel size should match.

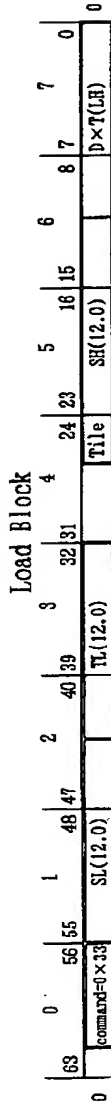
YUV SL must be even and SH must be odd, which means the width is even, to provide a UV value at each texel. For bilinear interpolation, tile SH should be adjusted to tile load SH-1, so that clamping occurs at a UV value.

For YUV and 32b RGBA images, software must specify a Tile Base address in the low half of Tmem, and load a tile which fits in the low half (where 2048<Base+WH in 8b Y's).

For YUV and 32b RGBA images, can't have color index TLUT's in Tmem.

Fractional coordinates for L and H terms must usually be the same. Useful for subpixel coordinate offset in multi-tile operations.

FIG. 70



Load Block

Field	Word	Bits	Description
Command	0	61-66	Command identifier. Load Block loads a Texen tile with a single memory "span" from SL, TL to SH, TL. During the tile load, the T coordinate is incremented by DxT every 8 Texen bytes, in order to perform odd-line swapping and line strides.
SL	0	55-64	Low S coordinate of tile in image
TL	0	43-52	Low T coordinate of tile in image, the two MSBs of this number should be zero (i.e., a 10-bit number)
Tile	0	28-34	Tile descriptor index
SH	0	23-32	High S coordinate of tile in image
DxT	0	11-20	unsigned T increment value

Load Block Usage Notes:

A ceiling function must be performed on the LH DxT field of Load Block. That is, if any number has non-zero bits to the right of the 11-bit fraction, then a ceiling operation should be performed on the number. For example, a 12 texel(16b/texel)wide texture would have a DxT of 1/3. The 11b fraction would be $1/3 \times 2048$ or 682 2/3. The ceiling is 683.

The texture image width must be multiples of 8 bytes. For example, a 4bit texel texture must have an image width of $n \times 16$.

Each Load Block command should be followed by Set Tile, Sz command to set the actual tile SL, TH values.

4b textures should be loaded as bytes(they must be byte aligned)using the Set Image Texture type field when loading more than 4k texels. This means the Load Block parameters will have 8b texel units. The Set Tile type field can be used to set the proper 4b type after the Load Block. The Set Tile Size can be used to set the proper SL, SH, TL, TH values after the Load Block. In general, textures can be loaded as one type and used as another type by proper manipulation of the Set Image Texture and Set Tile format size fields.

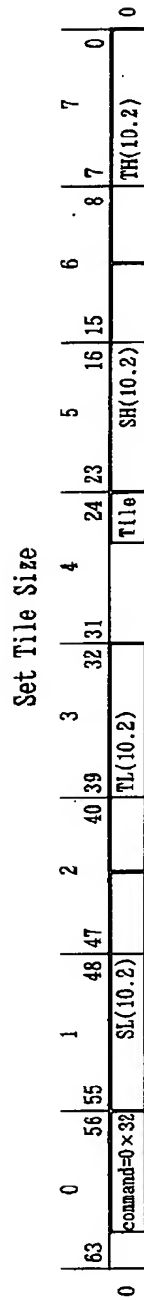
Normally, during a load, the tile texel size and image pixel size should match.

YUV texture images are stored as interleaved 8bit YUV as 16 bits per texel. Software must specify texture coordinates and tile coordinates and sizes which are even numbers in S(lab=0), so that a pair UV values are available. Software must specify a file base address in the low half of Texen, and load a tile which fits in the low half(where $2048 < base + W \times H$) in 8b Y's. Software can't have C1 TLUT's and YUV textures coexisting in Texen.

In Load Block, the Tile line is the number of words to skip for each T. That is, zero for a contiguous tile.

When Load Block is used to load multiple tiles of different widths(such as a mipmap pyramid), the image data in memory must be ordered for Texen interleaved access. This means that for odd lines(1&1), the two longs(32b) in each double(64b) must be swapped. Since Load Block lines must consist of an integral number of double words, this is not effected by width. Note that Load Tile performs this interleaved during load, and Load Block can perform this interleaved by computing a T coordinate from DxT in the command. For memory image data which is interleaved, DxT should be zero.

FIG. 71



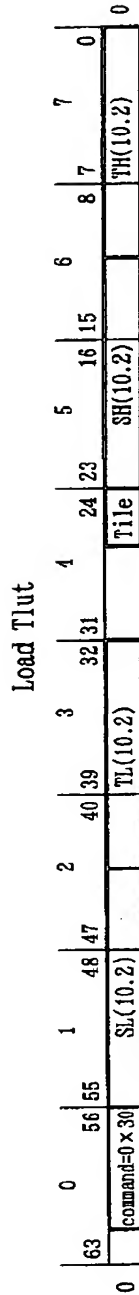
Set Tile Size

Field	Word	Bits	Description
command	0	61-56	command identifier
SL	0	55-44	Low S coordinate of tile in image
TL	0	43-32	Low T coordinate of tile in image
Tile	0	26-24	Tile descriptor index
SH	0	23-12	High S coordinate of tile in image
TH	0	11-0	High T coordinate of tile in image

Set Tile Size Usage Notes:

For YUV textures SL must be even and SH must be odd (this means the width is an even number of texels). Hardware will clamp to SH-1, that is, the last even texel, in order to have a valid UV at the last texel. If linear interpolation is performed, the max S texture coordinate must be odd, and SH is the max S+2 (an odd SH), in order to supply a valid UV at the max S+1.

FIG. 72



Load_Tlut

Field	Word	Bits	Description
command	0	61-56	command identifier, this command is used to initiate a load from DRAM of a Indexed Texture Lookup Table(TLUT). This table dereferences color indexed texels before texture filtering.
SL	0	55-44	low index into table(0-255), fractional bits should be zero
TL	0	43-32	normally zero
Tile	0	26-24	Tile descriptor index
SH	0	23-12	high index into table(0-255), fractional bits should be zero
TH	0	11-0	normally zero

Load_Tlut Usage Notes:

Use the set_image_texture to define the dram address, with a 16b type. Use set_tile to define the Tmem address. Reference that tile in load_tlut. The Texture Coordinate unit should get a DXS of 4 due to quadruplication of table when loading.

Tmem Address must be in high half(msb==1) of Tmem.

Tmem address in 64b words.

FIG. 73

Triangle command of various types are formed by concatenating groups of coefficients as shown below. The order for concatenation is from left to right in the table. The formats for each group of coefficients are shown on the following pages:

Triangle Commands

Command	Edge	Shade	Texture	ZBuffer
Non-Shaded Triangle, 0x08	✓			
Shade Triangle, 0x0c	✓	✓		
Texture Triangle, 0x0a	✓		✓	
Shade, Texture Triangle, 0x0e	✓	✓	✓	
Non-Shaded, ZBuff Triangle, 0x09	✓			✓
Shade, ZBuff Triangle, 0x0d	✓	✓		✓
Texture, ZBuff Triangle, 0x0b	✓		✓	✓
Shade, Texture, ZBuff Triangle, 0x0f	✓	✓	✓	✓

FIG. 74

Edge Coefficients

	63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0	command	4	Level	tile							YL(s,H.2)					YH(s,H.2)							YH(s,H.2)
1		XL									XL,frac					DxLDy							DxLDy,frac
2		XH									XH,frac					DxHDy							DxHDy,frac
3		XM									XM,frac					DxMDy							DxMDy,frac

Edge Coefficients

Field	Word	Bits	Description
command	0	61-56	Command identifier
lft	0	55	Left major flag, 1=left major, 0=right major
Level	0	53-51	number of mip-maps minus one
file	0	50-48	Tile descriptor index. Used to reference texture for this primitive.
YL	0	45-32	Y coordinate of low minor edge
YH	0	29-16	Y coordinate of mid minor edge
YH	0	13-0	Y coordinate of major edge
XL	1	63-48	X coordinate of low edge, integer
XL,frac	1	47-32	X coordinate of low edge, fraction
DxLDy	1	31-16	Inverse slope of low edge, integer
DxLDy,frac	1	15-0	Inverse slope of low edge, fraction

FIG. 75

Edge Coefficients

Field	Word	Bits	Description
XH	2	63-48	X coordinate of major edge, integer
XH, frac	2	47-32	X coordinate of major edge, fraction
DxHdy	2	31-16	Inverse slope of major edge, integer
DxHdy, frac	2	15-0	Inverse slope of major edge, fraction
XM	3	63-48	X coordinate of middle edge, integer
XM, frac	3	47-32	X coordinate of middle edge, fraction
DxMDy	3	31-16	Inverse slope of middle edge, integer
DxMDy, frac	3	15-0	Inverse slope of middle edge, fraction

FIG. 76

The edge coefficients are calculated at specific points on the view screen. The diagram below shows where each term is located. In general, y terms are calculated to at least subpixel($1/4$ pixel) resolution. The x_M and x_H are calculated where the H and M edges intersect the previous scan line. x_L is calculated where the L edge intersects the next subpixel at or below the mid vertex.

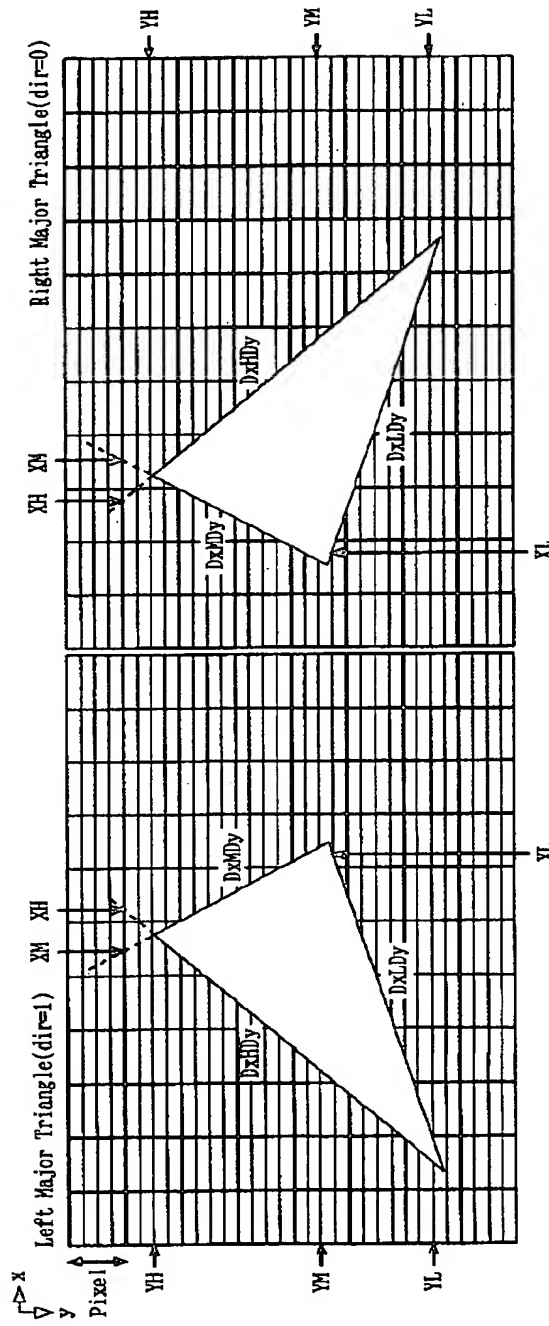


FIG. 77

Shade Coefficients																						
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
4		Red					Green						Blue						Alpha			
5		DrDx					DgDx						DbDx						DaDx			
6		Red Fraction					Green Fraction						Blue Fraction						Alpha Fraction			
7		DrDx, fraction					DgDx, fraction						DbDx, fraction						DaDx, fraction			
8		DrDe					DgDe						DbDe						DaDe			
9		DrDy					DgDy						DbDy						DaDy			
10		DrDe, fraction					DgDe, fraction						DbDe, fraction						DaDe, fraction			
11		DrDy, fraction					DgDy, fraction						DbDy, fraction						DaDy, fraction			

Shade Coefficients

Field	Word	Bits	Description
Red	4	63-48	Red color component, integer
Green	4	47-32	Green color component, integer
Blue	4	31-16	Blue color component, integer
Alpha	4	15-0	Alpha color component, integer
DrDx	5	63-48	Change in red per change in X coordinate, integer
DgDx	5	47-32	Change in green per change in X coordinate, integer

FIG. 78

Shade Coefficients

Field	Word	Bits	Description
DdDx	5	31-16	Change in blue per change in X coordinate, integer
DdDy	5	15-0	Change in alpha per change in X coordinate, integer
Red, frac	6	63-48	Red color component, fraction
Green, frac	6	47-32	Green color component, fraction
Blue, frac	6	31-16	Blue color component, fraction
Alpha, frac	6	15-0	Alpha color component, fraction
Drdx, frac	7	63-48	Change in red per change in X coordinate, fraction
Dgdx, frac	7	47-32	Change in green per change in X coordinate, fraction
Dbdx, frac	7	31-16	Change in blue per change in X coordinate, fraction
Dalx, frac	7	15-0	Change in alpha per change in X coordinate, fraction
Drdx	8	63-48	Change in red along the edge, integer
Dgdx	8	47-32	Change in green along the edge, integer
Dbdx	8	31-16	Change in blue along the edge, integer
Dadx	8	15-0	Change in alpha along the edge, integer
Drdy	9	63-48	Change in red per change in Y coordinate, integer
Dgdy	9	47-32	Change in green per change in Y coordinate, integer
Dbdy	9	31-16	Change in blue per change in Y coordinate, integer
Daldy	9	15-0	Change in alpha per change in Y coordinate, integer
Drdy	10	63-48	Change in red along the edge, fraction
Dgdy	10	47-32	Change in green along the edge, fraction
Dbdy	10	31-16	Change in blue along the edge, fraction
Daldy	10	15-0	Change in alpha along the edge, fraction
Drdy	11	63-48	Change in red per change in Y coordinate, fraction
Dgdy	11	47-32	Change in green per change in Y coordinate, fraction
Dbdy	11	31-16	Change in blue per change in Y coordinate, fraction
Daldy	11	15-0	Change in alpha per change in Y coordinate, fraction

FIG. 79

Texture Coefficients																						
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
12		S						T						W					Unused			
13		DsDx						DtDx						DwDx					Unused			
14		S Fraction						T Fraction						W Fraction					Unused			
15		DsDx, fraction						DtDx, fraction						DwDx, fraction					Unused			
16		DsDe						DtDe						DwDe					Unused			
17		DsDy						DtDy						DwDy					Unused			
18		DsDe, fraction						DtDe, fraction						DwDe, fraction					Unused			
19		DsDy, fraction						DtDy, fraction						DwDy, fraction					Unused			

Texture Coefficients

Field	Word	Bits	Description
S	12	63-48	S texture coordinate, integer
T	12	47-32	T texture coordinate, integer
W	12	31-16	Normalized inverse depth, integer
DsDx	13	63-48	Change in S per change in X coordinate, integer
DtDx	13	47-32	Change in T per change in X coordinate, integer
DwDx	13	31-16	Change in W per change in X coordinate, integer

FIG. 80

Texture Coefficients

Field	Word	Bits	Description
S, frac	14	63-48	S texture coordinate, fraction
T, frac	14	47-32	T texture coordinate, fraction
W, frac	14	31-16	Normalized inverse depth, fraction
Dsdx, frac	15	63-48	Change in S per change in X coordinate, fraction
Dtdx, frac	15	47-32	Change in T per change in X coordinate, fraction
Dwdx, frac	15	31-16	Change in W per change in X coordinate, fraction
Dsde	16	63-48	Change in S along the edge, integer
Dtde	16	47-32	Change in T along the edge, integer
Dwde	16	31-16	Change in W along the edge, integer
Dsdy	17	63-48	Change in S per change in Y coordinate, integer
Dtdy	17	47-32	Change in T per change in Y coordinate, integer
Dwdy	17	31-16	Change in W per change in Y coordinate, integer
Dsde, frac	18	63-48	Change in S along the edge, fraction
Dtde, frac	18	47-32	Change in T along the edge, fraction
Dwde, frac	18	31-16	Change in W along the edge, fraction
Dsdy, frac	19	63-48	Change in S per change in Y coordinate, fraction
Dtdy, frac	19	47-32	Change in T per change in Y coordinate, fraction
Dwdy, frac	19	31-16	Change in W per change in Y coordinate, fraction

2Buffer Coefficients

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
63	56	55	48	47	40	39	32	31	24	23	16	15	8	7								
	z				z, frac				DzDx				DzDx, frac									
20																						
21	DzDe				DzDe, frac				DzDy				DzDy, frac									

zBuffer Coefficients

Field	Word	Bits	Description
Z	20	63-48	Inverse Depth, integer
Z, frac	20	47-32	Inverse Depth, fraction
DzDx	20	31-16	Change in Z per change in X coordinate, integer
DzDx, frac	20	15-0	Change in Z per change in X coordinate, fraction
DzDe	21	63-48	Change in Z along major edge, integer
DzDe, frac	21	47-32	Change in Z along major edge, fraction
DzDy	21	31-16	Change in Z per change in Y coordinate, integer
DzDy, frac	23	15-0	Change in Z per change in Y coordinate, fraction

FIG. 82

Fill Rectangle																							
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0	
command=0x36				XL(10.2)				YL(10.2)				XH(10.2)				YH(10.2)				0			

Fill_Rectangle

Field	Word	Bits	Description
command	0	61-56	Command identifier
XL	0	55-44	X coordinate of bottom right corner of rectangle
YL	0	43-32	Y coordinate of bottom right corner of rectangle
XH	0	23-12	X coordinate of top left corner of rectangle
YH	0	11-0	Y coordinate of top left corner of rectangle

FIG. 83

Texture Rectangle																										
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0				
0	command=0x24				XL(10.2)				YL(10.2)				tile				XH(10.2)				YH(10.2)				0	
1	S(s,0.5)				T(s,0.5)				DsDx(s,5.10)				DsDy(s,5.10)										1			

Texture Rectangle

Field	Word	Bits	Description
command	0	61-56	Command identifier
XH	0	55-44	X coordinate of top left corner of rectangle
YH	0	43-32	Y coordinate of top left corner of rectangle
tile	0	26-24	Title descriptor index
XL	0	23-12	X coordinate of bottom right corner of rectangle
YL	0	11-0	Y coordinate of bottom right corner of rectangle
S	1	63-48	S texture coordinate at top left corner of rectangle
T	1	47-32	T texture coordinate at top left corner of rectangle
DsDx	1	31-16	Change in S per change in X coordinate
DtDy	1	15-0	Change in T per change in Y coordinate

Texture Rectangle Usage Notes:

To copy an image, set cycle_type to "copy" in Set_Other_Modes, and set the combination of Set_Tile's "shift S" and DsDx to step by 4 texels. No texture filtering, color combining, or blending operations are available for copied texels. Write enables may be generated using threshold compares of the alpha channel of each copied texel.

Texture Rectangle and Copy mode:

No Z-buffer or anti-aliasing in copy mode.

4b, YUV, and 32b RGBA textures cannot be directly copied.

4b images are expanded to 8b images before copying.

4b and 8b images can be copied to an 8b color image only.

16b image can be copied to a 16b color image only.

Note that 4b, YUV, and 32b RGBA images can be copied by using 8b or 16b color images and correctly scaling image coordinates and width.

FIG. 84

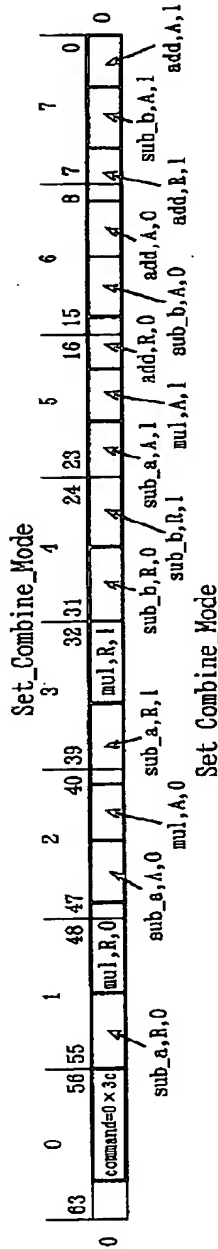
Texture Rectangle Flip

	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
0		command=0x25		XL(10.2)						YL(10.2)				tile		XH(10.2)						YH(10.2)
1		S(s,0.5)					T(s,0.5)							DsDx(s,5,10)								DtDy(s,5,10)

Texture_Rectangle Flip

Field	Word	Bits	Description
command	0	61-56	Command identifier, same as Texture Rectangle except hardware swaps S/T and DsDx/DtDy.
XH	0	55-44	X coordinate of top left corner of rectangle
YH	0	43-32	Y coordinate of top left corner of rectangle
tile	0	26-24	Tile descriptor index
XL	0	23-12	X coordinate of bottom right corner of rectangle
YL	0	11-0	Y coordinate of bottom right corner of rectangle
S	1	63-48	S texture coordinate at top left corner of rectangle
T	1	47-32	T texture coordinate at top left corner of rectangle
DsDx	1	31-16	Change in S per change in X coordinate
DtDy	1	15-0	Change in T per change in Y coordinate

FIG. 85

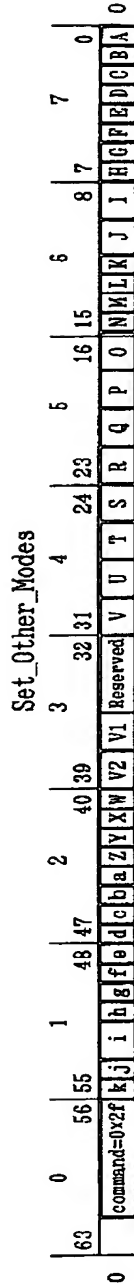


Field	Word	Bits	Description
command	0	61-56	Command identifier
sub_a,R,0	0	55-52	sub_A input, RGB components, cycle 0
mul,R,0	0	51-47	multiply input, RGB components, cycle 0
sub_a,A,0	0	46-44	sub_A input, Alpha component, cycle 0
mul,A,0	0	43-41	multiply input, Alpha component, cycle 0
sub_a,R,1	0	40-37	sub_A input, RGB components, cycle 1
mul,R,1	0	36-32	multiply input, RGB components, cycle 1
sub_b,R,0	0	31-28	sub_B input, RGB components, cycle 0
sub_b,R,1	0	27-24	sub_B input, RGB components, cycle 1
sub_a,A,1	0	23-21	sub_A input, Alpha component, cycle 1
mul,A,1	0	20-18	multiply input, Alpha component, cycle 1
add,R,0	0	17-15	adder input, RGB components, cycle 0
sub_b,A,0	0	14-12	sub_B input, Alpha component, cycle 0
add,A,0	0	11-9	adder input, Alpha components, cycle 0
add,R,1	0	8-6	adder input, RGB components, cycle 1
sub_b,A,1	0	5-3	sub_B input, Alpha components, cycle 1
add,A,1	0	2-0	adder input, Alpha components, cycle 1

Set_Combine_Mode Usage Notes:

The Color Combiner implements the equation: $(A-B)^{C+D}$ on each color. RGB and Alpha channels have separate mux selects. In addition, there are separate mux selects for cycle 0 and cycle 1. If the RDP is configured for one cycle mode, set the cycle 0 and cycle 1 mux selects to the same value.

FIG. 86



Set_Other_Modes

Field	Word	Bits	Description
command	0	61:56	Command identifier
k:atomic_prim	0	55	Force primitive to be written to frame buffer before read of following primitive.
j:Reserved	0	54	This mode bit is not currently used but may be in the future.
i:cycle_type	0	53:52	Display pipeline cycle control mode:0=1 cycle, 1=2 cycle, 2=Copy, 3=Fill.
h:persp_tex_en	0	51	enable perspective correction on texture
g:detail_tex_en	0	50	enable detail texture
f:sharpen_tex_en	0	49	enable sharpened texture
e:tex_lod_en	0	48	enable texture Level of Detail (LOD)
d:en_tlut	0	47	enable lookup of texel values from TLUT. Meaningful if texture type is Index, tile is in low Them, TLUT is in high Them, and color image is RGB.
c:tlut_type	0	46	Type of texels in table, 0=16b RGBA(5/5/5/1), 1=1A(8/8)
b:sample_type	0	45	determines how textures are sampled: 0=1x1(Point Sample), 1=2x2. Note that copy (point sample 4 horizontally adjacent texels) mode is indicated by cycle_type.
a:mid_texel	0	44	indicates texture filter should do a 2x2 half texel interpolation, primarily used for MPEG motion compensation processing.
Z:bi_lerp_0	0	43	1=bi_lerp, 0=color convert operation in texture filter. Used in cycle 0
Y:bi_lerp_1	0	42	1=bi_lerp, 0=color convert operation in texture filter. Used in cycle 1
X:convert_one	0	41	Color convert texel that was the output of the texture filter on cycle 0, used to qualify bi_lerp_1
W:key_en	0	40	Enables chroma keying

FIG. 87

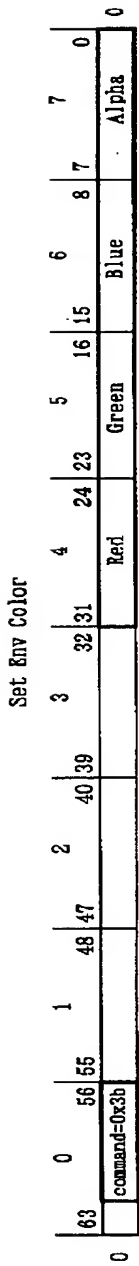
Set_Other_Modes

Field	Word	Bits	Description
V2:rgb_dither_sel	0	39:38	0=magic square matrix(preferred if filtered) 1=standard Bayer matrix(preferred if not filtered) 2=noise(as before) 3=no dither
V1:alpha_dither_sel	0	37:36	0=pattern 1=pattern 2=noise 3=no dither
Reserved	0	35:32	Reserved for future use, default value is 0xf
V:b,alb,0	0	31:30	Blend mode word, multiply 1a input select, cycle 0
U:b,alb,1	0	29:28	Blend mode word, multiply 1a input select, cycle 1
T:b,alb,0	0	27:26	Blend mode word, multiply 1b input select, cycle 0
S:b,alb,1	0	25:24	Blend mode word, multiply 1b input select, cycle 1
R:b,a2a,0	0	23:22	Blend mode word, multiply 2a input select, cycle 0
Q:b,a2a,1	0	21:20	Blend mode word, multiply 2a input select, cycle 1
P:b,a2b,0	0	19:18	Blend mode word, multiply 2b input select, cycle 0
O:b,a2b,1	0	17:16	Blend mode word, multiply 2b input select, cycle 1
N:reserved	0	15	This mode bit is not currently used, but may be in the future
M:force blend	0	14	force blend enable
L:alpha_cvg_select	0	13	use cvg(or cvga) for pixel alpha
K:cvg_times_alpha	0	12	use cvg times alpha for pixel alpha and coverage
J:z_mode[1:0]	0	11:10	0:opaque, 1:interpenetrating, 2:transparent, 3:ideal
I:avg_dest[1:0]	0	8:9	0:clamp(normal), 1:wrap(as assume fill cvg), 2:wrap(force to full cvg), 3:save(don't overwrite memory cvg).
H:color_on_cvg	0	7	only update color on coverage overflow(transparent surfaces)
G:image_read_en	0	6	enable color/cvg read/modify/write memory access
F:z_update_en	0	5	enable writing of z if color write enabled
E:z_compare_en	0	4	conditional color write enable on depth comparison
D:antialias_en	0	3	if not force blend, allow blend enable-use cvg bits
C:z_source_sel	0	2	choose between Primitive z and pixel z
B:dither_alpha_en	0	1	use random noise in alpha compare, otherwise use blend alpha in alpha compare
A:alpha_compare_en	0	0	conditional color write on alpha compare

Set_Other_Modes Usage Notes:

VUV Copy mode is not supported.
Fill mode in "Cycle Type" means replicate the Fill Color(see Set Fill Color).
Initialize the Z-buffer by setting the color image to point to the z-buffer(Set Color Image) and filling with initial depth value.
Multi-tile mixed color index thru TOUT and other not supported, because TOUT effects all modes.

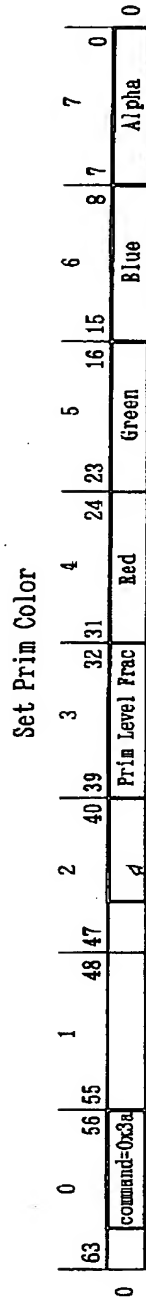
FIG. 88



Set Env Color

Field	Word	Bits	Description
command	0	61-56	Command identifier
Red	0	31-24	Red Component
Green	0	23-16	Green Component
Blue	0	15-8	Blue Component
Alpha	0	7-0	Alpha Component

FIG. 89



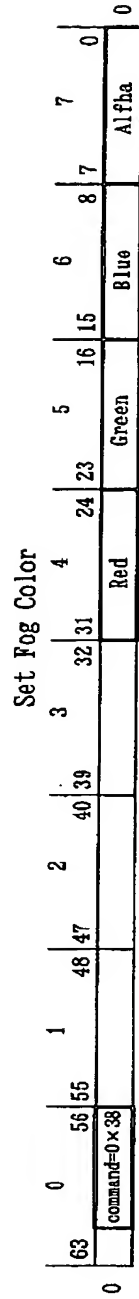
Set Env Color

Field	Word	Bits	Description
command	0	61-56	Command identifier
Prim Min Level	0	44-40	Minimum clamp for LOD fraction when in detail or sharpen texture modes, fixed point 0.5.
Prim Level Frac	0	39-32	Level of Detail fraction for primitive, used primarily in multi-tile operations for rectangle primitives, 0.8.
Green	0	23-16	Green Component
Blue	0	15-8	Blue Component
Alpha	0	7-0	Alpha Component

FIG. 90

Set Blend Color																											
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0					
command=0x39												Red				Green				Blue				Alpha			

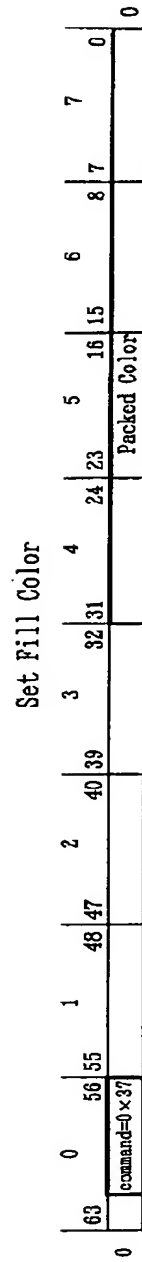
FIG. 91



Set Fog Color

Field	Word	Bits	Description
command	0	61-56	Command identifier
Red	0	31-24	Red Component
Green	0	23-16	Green Component
Blue	0	15-8	Blue Component
Alpha	0	7-0	Alpha Component

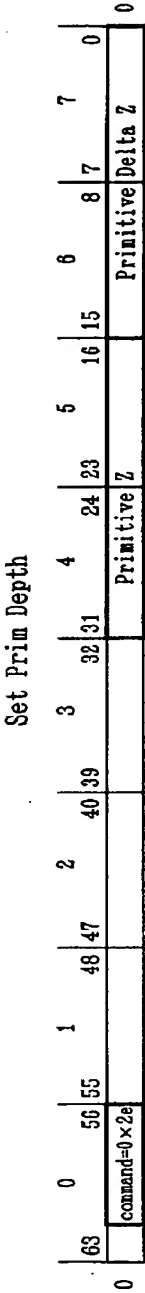
FIG. 92



Set Fill Color

Field	Word	Bits	Description
command	0	61-56	Command identifier
Packed Color	0	31-0	Packed Color. For example, if the Color image was set be 16b RGBA, then the fill color would be two horizontally adjacent 16b RGBA pixels.

FIG. 93



Set Prim Depth

Field	Word	Bits	Description
command	0	61-56	Command Identifier
Primitive Z	0	31-16	Primitive Z
Primitive Delta Z	0	15-0	Primitive Delta Z

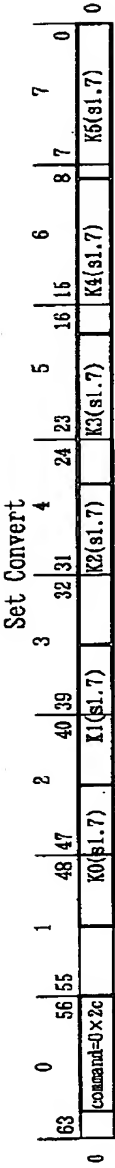
FIG. 94

Set Scissor																						
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
command=0x2d				XH(10.2)						YH(10.2)				f0		XL(10.2)				YL(10.2)		0

Set Scissor

Field	Word	Bits	Description
command	0	61-56	Command identifier
XH	0	55-44	X coordinate of top left corner of scissor box in screen space.
YH	0	43-32	Y coordinate of top left corner of scissor box in screen space.
f	0	25	scissor field, enables scissoring of odd or even lines for interlaced displays
0	0	24	odd line:0=keep even line, 1=keep odd line, indicates whether all odd lines or all even lines should be skipped(for interlaced displays).
XL	0	23-12	X coordinate of bottom right corner of scissor box in screen space.
YL	0	11-0	Y coordinate of bottom right corner of scissor box in screen space.

FIG. 95



Set Convert

Field	Word	Bits	Description
command	0	61-56	Command identifier, this command updates the coefficients for converting YUV pixels to RGB. Conceptually the equations are: $R=C0^*(Y-16)+C1^*V$ $G=C0^*(Y-16)-C2^*U-C3^*V$ $B=C0^*(Y-16)+C4^*U$
K0	0	53-45	K0 term of YUV-RGB conversion matrix
K1	0	44-36	K1 term of YUV-RGB conversion matrix
K2	0	35-27	K2 term of YUV-RGB conversion matrix
K3	0	26-18	K3 term of YUV-RGB conversion matrix
K4	0	17-9	K4 term of YUV-RGB conversion matrix
K5	0	8-0	K5 term of YUV-RGB conversion matrix

Set_Convert Usage Notes:

In the hardware, the color conversion is done in two stages.
In the texture filter(TF), the following equation is performed:

$R'=Y+K0*V$
 $G'=Y+K1*U+K2*V$
 $B'=Y+K3*U$

In the color combiner, the following equations are performed:

$R=(R'-K4)*K5+R'$
 $G=(G'-K4)*K5+G'$
 $B=(B'-K4)*K5+B'$

Where(CX terms as defined in the table above):

- K0=C1/C0
- K1=C2/C0
- K2=C3/C0
- K3=C4/C0
- K4=16+16/(C0-1.0)
- K5=C0-1.0

Typical Values for YUV to RGB conversion:

- K0=175
- K1=-43
- K2=-89
- K3=222
- K4=114
- K5=42

FIG. 96

Set Key R																									
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0			
command=0x2b														Width R(s7.4)					Center R			0			
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0
																									0

FIG. 97

Set Key GB																											
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0					
command=0x2a				Width G				Width B				Center G				Scale G				Center B				Scale B			
Set_Key_GB																											
Field				Word				Bits				Description															
command				0				61-56				Command identifier, This command set the coefficients used for Green/Blue keying. Conceptually, the equation used for keying is: $\text{KeyG/B} = \text{clamp}(0, 0, -\text{abs}(\text{G/B} - \text{Center}) * \text{Scale}) * \text{Width}, 1, 0).$ The Key Alpha is the minimum of the KeyR, KeyG, KeyB.															
Width G				0				55-44				(Size of half the key window including the soft edge)*scale. If width>1.0, then keying is disabled for that channel.															
Width B				0				43-32				(Size of half the key window including the soft edge)*scale. If width>1.0, then keying is disabled for that channel.															
Center G				0				31-24				Defines color or intensity at which key is active, 0-255															
Scale G				0				23-16				(1.0/(size of soft edge)). For hard edge keying, set scale to 255.															
Center B				0				15-8				Defines color or intensity at which key is active, 0-255															
Scale B				0				7-0				(1.0/(size of soft edge)). For hard edge keying, set scale to 255.															

Set_Key_XX Usage Notes:

In the hardware, the keying equation is performed in two stages.

In the Color Combiner(CC), the equation performed is:

$$\text{Key}' = (\text{pixel} - \text{Center}) * \text{Scale} + 0$$

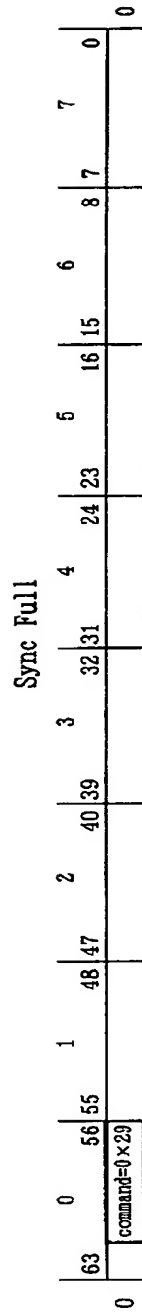
In the Alpha Fixup unit(AP), the equation performed is:

$$\text{Key} = \text{clamp}(0, -\text{abs}(\text{Key}') + \text{Width}, 1, 0)$$

$$\text{KeyAlpha} = \text{MIN}(\text{KeyB}, \text{KeyG}, \text{KeyB})$$

In two-cycle mode, the keying operation must be specified in the second cycle (that is, the key alpha is not available as a combine operand).

FIG. 98



Sync Full

Field	Word	Bits	Description
command	0	61-56	Command identifier. This command stalls the RDP until the last dram buffer is read or written from any preceding primitive. It is typically only needed if the memory data is to be reused, like switching display buffers, or writing a color_image to be used as a texture_image, or for consistent r/w access to an RDP w/r image from the cpu.

FIG. 99

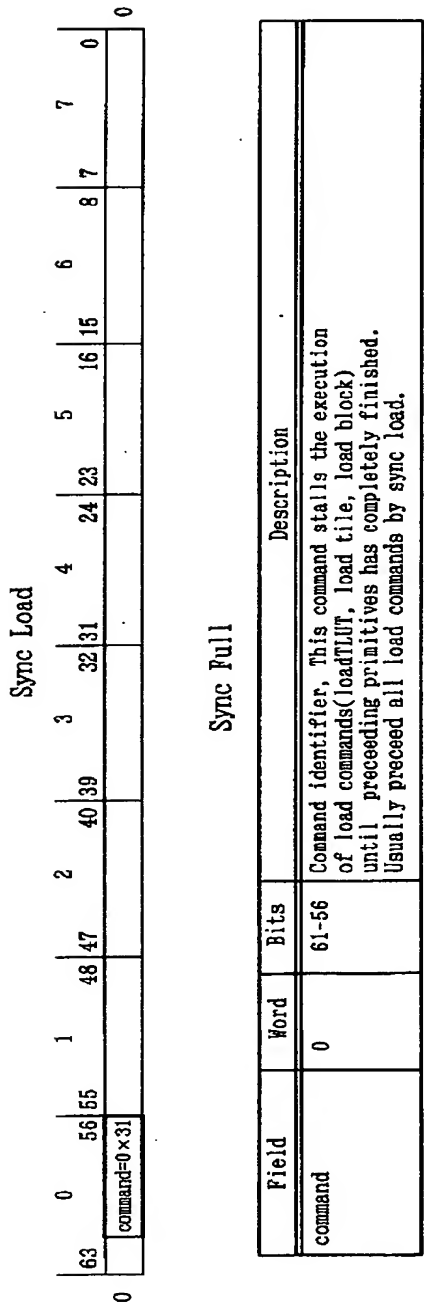
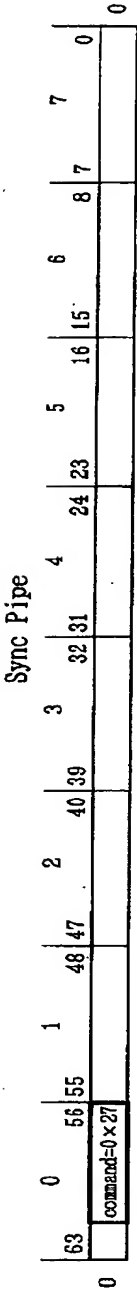


FIG. 100



Sync Pipe

Field	Word	Bits	Description
command	0	61-56	Command identifier, General attributes (other than prim color/depth) which are being read by up to two preceding primitives should be preceded by sync_pipe, which stalls until the most recent primitive is past the last usage of any attribute. Only one sync_pipe is needed before any number of attribute commands. Software can optimize sync_pipe usage if it knows what is being read, for example, a set_texture_image can follow tri's or rects in preparation for a load_tile without a sync_pipe, because tris or rects don't use the texture_image attribute. In general, the RDP is optimized for a number of primitives rendered with the same attribute setting. If attributes change per primitive, performance will degrade slightly.

FIG. 101

Sync Tile																							
63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0	
command=0x28																							0

Sync Tile

Field	Word	Bits	Description
command	0	61-56	Command identifier, Allows synchronization between commands that write to the same tile descriptor that an immediately previous command is reading.

FIG. 102

63	0	56	55	1	48	47	2	40	39	3	32	31	4	24	23	5	16	15	6	8	7	0
No Op																						
0																						
command=0x00																						

No Op

Field	Word	Bits	Description
command	0	61-56	Command identifier. This command has no effect on rdp command execution but is useful for padding command buffers.

1. Abstract

This invention is a low cost high performance 3D graphics system, within a low cost range that most consumers can afford, can model a world in 3D and project the model onto a two dimensional viewing plane selected based on a changeable viewpoint. The viewpoint can be changed on an interactive, real time basis by controlling user input operations by game controllers and so forth. The system rapidly produces a corresponding changing image on the screen of a color television set.

This invention describes a structure and operation of a graphics/audio coprocessor for implementing the above-mentioned features, more specifically, an internal structure of the coprocessor, processings performed for a graphics/audio output, a method of communicating with the external world, a unified RAM and commands and associated formats sent by a main processor to the coprocessor for controlling the coprocessor.

2. Representative Drawing

Fig. 1